SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Jakov Topić

Zagreb, 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Prof. dr. sc. Joško Deur, dipl. ing.

Student:

Jakov Topić

Zagreb, 2017.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru prof.dr.sc Jošku Deuru na suradnji, korisnim savjetima i vodstvu koje mi je pružao za vrijeme pisanja ovog rada.

Također se zahvaljujem dr.sc. Branimiru Škugoru na izrazito velikoj stručnoj pomoći, strpljenju, savjetima i danoj stručnoj literaturi koja mi je znatno pomogla u pisanju rada.

Na kraju bih se želio duboko zahvaliti svojoj obitelji, svim prijateljima i svojoj djevojci Karmeli na povjerenju i moralnoj podršci koju su mi pružali tijekom studija.

Jakov Topić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **Jakov Topić** Mat. br.: 0152195996

Naslov rada na hrvatskom jeziku: **Izrada programske aplikacije za simulaciju sustava dijeljenja električnih mopeda**

Naslov rada na engleskom jeziku: **Design of a software application for simulation of an electric scooter sharing system**

Opis zadatka:

U gradovima raste trend uvođenja sustava dijeljenja vozila, kako bi se poboljšala mobilnost građana te smanjili problemi s prometnim gužvama i parkingom. Pritom se naglasak stavlja na električna vozila zbog smanjenja zagađenja okoliša i buke. Kako bi se ostvarila puna funkcionalnost sustava dijeljenja vozila, potrebno je koristiti napredne informacijske i komunikacijske tehnologije (ICT) u upravljanju i nadzoru sustava. Cjeloviti sustav uključuje centar vođenja, sustav punionica električnih vozila te sama električna vozila s uključenim modulom za GPS/GSM praćenje vozila. U radu je za sustav dijeljenja električnih mopeda i primjer grada Dubrovnika potrebno:

- Izraditi računalni program koji emulira rad centra vođenja, a koji će podatke pristigle u realnom vremenu s e-mopeda i punionica pohranjivati u bazu podataka i na temelju njih dodjeljivati mopede korisnicima;
- izraditi računalni program koji emulira rad sustava upravljanja punionicama u realnom vremenu;
- izraditi simulacijsko okruženje u sklopu kojeg će se ispitivati funkcionalnost razvijenih računalnih programa, a koje će sadržavati simulaciju 24-satnih prijava korisnika na pojedine punionice i odabira destinacija, te simulaciju modela e-mopeda uz primjenu realističnih voznih ciklusa;
- izraditi računalni program koji će na temelju matrica prijelaznih vjerojatnosti, parametriranih podacima snimljenim na stvarnim vozilima, generirati sintetičke vozne cikluse koji statistički vjerno reprezentiraju snimljene vozne cikluse.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
17. studenog 2016.	19. siječnja 2017.	25., 26. i 27. siječnja 2017.

Zadatak zadao: v. d. predsjednika Povjerenstva:

Prof. dr. sc. Joško Deur Prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA I KRATICA

Popis oznaka:

Oznaka	Jedinica	Opis
a	m/s^2	Akceleracija
A_f	m^2	Površina frontalnog presjeka vozila
c_1, c_2, c_3	-	Parametri aproksimacije ovisni o brzini vrtnje
C_d	-	Koeficijent aerodinamičkog otpora
e	m/s	Pogreška regulacije
E_{tot}	kWh	Ukupna utrošena električna energija
$E_{\max}$	kWh	Energetski kapacitet baterije
f_w	-	Vremenski prozor unutar kojeg se obavlja optimiranje
F	-	Funkcija cilja
g	m/s^2	Gravitacijska konstanta
i_o	-	Prijenosni omjer remenice
K_p	-	Pojačanje procesa
m_v	kg	Ukupna masa vozila koja uključuje i masu vozača
$m_{v,praznog}$	kg	Masa praznog vozila
$m_{v,putnika}$	kg	Procijenjena masa vozača
M	-	Broj uzoraka za optimiranje funkcije cilja
N	-	Normalna razdioba
OBK	-	Očekivani broj korisnika
p_{ij}	-	Uvjetna vjerojatnost da će se sustav naći u *j*-tom stanju ako se prethodno nalazio u *i*-tom
$\boldsymbol{P}$	-	Matrica prijelaznih vjerojatnosti
P_{meh}	W	Mehanička snaga
P_{el}	W	Snaga električnog stroja
P_{bat}	W	Snaga baterije
$P_{mg,gub}$	W	Gubici snage električnog stroja
P_{mg,gub_min}	W	Minimalni gubici snage električnog stroja
r	m	Efektivni radijus gume
R	Ω	Unutarnji otpor baterije
R_o	-	Koeficijent trenja kotrljanja guma
s	m	Put
s_{uk}	m	Ukupno prijeđeni put
s_0	m	Trenutno prijeđeni put
s_{fc}, s_{mg}	-	Faktori skaliranja elektromotora
SoC	-	Stanje napunjenosti baterije (*engl. state of charge*)
SoH	-	Stanje zdravlja baterije (*engl. state of health*)

$SoC_{poč}$	-	Vrijednost stanja napunjenosti baterije na početku voznog ciklusa
SoC_{kraj}	-	Vrijednost stanja napunjenosti baterije na kraju voznog ciklusa
$t_{sim,tren}$	s	Trenutno vrijeme simulacije
$t_{sim,kraj}$	s	Vrijeme kraja simulacije
$t_{pristupa}$	s	Vrijeme pristupa korisnika na stanicu
T	s	Vremenski period
$Tr_{uk,el}$	kn	Ukupni troškovi električne energije
$Tr_{uk,fg}$	kn	Ukupni troškovi fosilnog goriva
u_0	-	Stacionarna vrijednost upravljačkog signala
U_{oc}	V	Napon otvorenog kruga baterije (*engl. open circuit voltage*)
v_v	m/s	Brzina vozila
$v_{v,sk}$	m/s	Skalirana brzina vozila
w	-	Težinski faktor stanice
$Q_{\max}$	As	Ukupni nabojski kapacitet baterije
α	rad	Nagib ceste
ρ	-	Težinski faktor funkcije cilja
ρ_{zraka}	kg/m^3	Gustoća zraka
η_t	-	Stupanj korisnosti mehaničke transmisije
λ	rad	Geografska širina (*engl. latitude*)
φ	rad	Geografska dužina (*engl. longitude*)
μ	-	Aritmetička sredina
σ	-	Standardno odstupanje
ω_L	rad/s	Brzina vrtnje kotača vozila
ω_{mg}	rad/s	Brzina vrtnje električnog stroja
τ_a	Nm	Moment ubrzanja vozila
τ_L	Nm	Ukupni moment na kotaču vozila
τ_{mg}	Nm	Moment električnog stroja
$\tau_{mg,\max}$	Nm	Maksimalni moment el. stroja
τ_{gub}	Nm	Ukupni gubici momenta vozila
τ_{aero}	Nm	Moment aerodinamičkog otpora
τ_{kot}	Nm	Moment otpora trenja kotrljanja
τ_{alfa}	Nm	Moment za svladavanje nagiba
ΔSoC	-	Vrijednost razlike stanja napunjenosti baterije s početka i kraja voznog ciklusa
ΔT_{sim}	s	Period uzorkovanja simulacije

Popis kratica:

Kratica	Opis
AJAX	Set tehnika za izradu asinkronih web aplikacija (*engl. Asynchronous JavaScript and XML*)
API	Sučelje za programiranje aplikacija (*engl. Application Programming Interface*)
CAN	Standardni komunikacijski protokol za vozila (*engl. Controller Area Network*)
CSS	Stilski jezik za opis HTML dokumenta (*engl. Cascading Style Sheet*)
DAL	Web2py API koji preslikava Python objekte u objekte baze podataka (*engl. Database Abstraction Layer*)
EVs	Električna vozila (*engl. Electric Vehicles*)
FPS	Broj slika u sekundi (*engl.Frames Per Second*)
GPRS	Paketna, bežična podatkovna komunikacijska usluga (*engl. General Packet Radio Service*)
GPS	Sustav za globalno pozicioniranje (*engl. Global Positioning System*)
GSM	Globalni sistem za mobilnu komunikaciju (*engl. Global System for Mobile Communications*)
GUI	Grafičko korisničko sučelje (*engl. Graphical User Interface*)
HMI	Sučelje čovjek-stroj (*engl. Human Machine Interface*)
HTTP	glavna i najčešća metoda prijenosa informacija na web-u (*engl. HyperText Transfer Protocol*)
IDE	Integrirano razvojno okruženje (*engl. Integrated Development Environment*)
IPv4	Najraširenija verzija Internet protokola (*engl. Internet Protocol version 4*)
JSON	Jednostavan format za razmjenu podataka (*engl. JavaScript Object Notation*)
MVC	Obrazac softverske arhitekture, model-pogled-kontroler (*engl. Model-View-Controller*)
PaaS	Kategorija usluge računarstva u oblacima - platforma kao usluga (*engl. Platform as a Service*)
RFID	Tehnologija koja koristi radio frekvenciju kako bi se razmjenjivale informacije između prijenosnih uređaja/memorija i host računala (*engl. Radio-Frequency Identification*)
SDL	Multimedijska programska knjižnica (*engl. Simple DirectMedia Layer*)
SQL	Najpopularniji računalni jezik za manipulaciju podacima iz relacijskih baza podataka (*engl. Structured Query Language*)
UID	Jedinstveni identifikator (*engl. Unique Identifier*)
URL	Ujednačeni lokator sadržaja - web adresa (*engl. Uniform Resource Locator*)
XML	jezik dizajniran za pohranu i transport podataka (*engl. eXtensible Markup Language*)

SAŽETAK

U ovom radu detaljno su opisani princip rada i struktura sustava za dijeljenje električnih mopeda za primjer grada Dubrovnika, zajedno s predloženim softverskim rješenjima centra vođenja i 24-satnog simulatora klijenta. Najprije je izvršeno rašlanjivanje procesa preuzimanja i vraćanja e-mopeda u korake kako bi se dobio što bolji uvid u zbivanja, tipove i tokove podataka. Zatim je u detalje opisana funkcionalnost mrežne aplikacije centra vođenja, kojom se nadzire i upravlja cjelokupnim sustavom, zajedno s pripadnim joj tehnologijama i razvojnim okruženjem koji su bili korišteni prilikom njene izrade. Nakon toga, postavljen je kvazistatički model e-mopeda u Matlab-Simulink okruženju, koji omogućuje generiranje ključnih varijabli mopeda bitnih za njegovo praćenje. Za potrebe što realističnijeg simuliranja sustava također su izvršena razna skaliranja i korekcije snimljenih podataka kako bi određena ograničenja (momenti, brzine, itd.) e-mopeda bila zadovoljena. Nadalje, za potrebe 24-satne simulacije sustava izrađena je i detaljno opisana aplikacija simulatora klijenta koja se sastoji od stohastičkih simulacija prijava korisnika na stanicu i određivanja sljedeće destinacije, generiranja sintetičkih voznih ciklusa primjenom Markovljevih lanaca, te animacije cjelokupnog sustava i prikaza dobivenih rezultata. Na kraju rada izvršena je analiza intenziteta posjećenosti svih stanica, zajedno s usporedbom procijenjenih ukupnih dnevnih i mjesečnih troškova hipotetskih flota električnih i konvencionalnih mopeda.

Ključne riječi: sustav dijeljenja, električni moped, flota vozila, nadzor sustava, centar vođenja, stohastičko simuliranje, kvazistatički model, Markovljevi lanci, sintetički vozni ciklusi, programska aplikacija

SUMMARY

In this thesis, the operating principle and structure of the electric moped sharing system for the case of the city of Dubrovnik, are described along with the proposed software solutions of control center and 24-hour client simulator. First, in order to obtain better insight into the events, types and flows of data, the detailed specification of e-moped pickup and return processes is given. Then, the functionality of the control center network application, which monitors and controls the entire system, is described in details, along with the development tools used for its realization. Afterwards, a quasistatic model of e-moped is built-up in the Matlab-Simulink environment, along with various scalings and corrections of recorded data in order to satisfy certain limitations (torque, speed, etc.). Furthermore, for the purpose of the 24-hour system simulation, a desktop aplication of the client simulator including stochastic models of user logins to the station and determining the next destination, generation of synthetic driving cycles using Markov chains, animation of the overall system together with visualisation of the obtained results, is developed and described in details. Finally, an analysis of the charging stations usage statistics is given, together with a comparative analysis of the estimated total daily and monthly energy costs for the case of hypothetical electrical and conventional moped fleets.

Key words: sharing system, electric moped, vehicle fleet, system supervision, control center, stochastic simulation, quasistatic model, Markov chains, synthetic driving cycles, software application

1. UVOD

1.1. Općenito o sustavima dijeljenja vozila

Zbog ekoloških zahtjeva modernog vremena, u urbanoj mobilnosti sve više raste trend razvoja transportnih sustava koji uključuju električna vozila. Osim toga potiču se i "zelenije" vrste transporta, kao što su hodanje, biciklizam i javni prijevoz. S ciljem poboljšanja načina prijevoza i smanjenja njegova utjecaja na okoliš, u posljednjih nekoliko godina sve veća pozornost posvećuje se i programima za dijeljenje vozila (bicikala, mopeda i automobila)[1].

Bike-sharing programi predstavljaju mreže bicikala namijenjenih za javnu uporabu, raspoređenih po čitavoj površini grada te dostupnih korisniku za uporabu uz niske troškove. Programi obuhvaćaju kratkoročne urbane sheme za najam bicikala koje omogućuju korisnicima da ih preuzmu s bilo koje stanice za bicikle, te da ih zatim vrate na bilo koju stanicu za bicikle, što ih čini idealnim za *point-to-point* putovanja. Načelo dijeljenja bicikala vrlo je jednostavno: pojedinci koriste bicikle po potrebi, bez troškova i odgovornosti vlasništva. Najraniji poznati društveni program bicikala pokrenut je 1965. godine u Amsterdamu u Nizozemskoj.

Trenutni *bike-sharing* sustavi podržavaju preuzimanje i vraćanje bicikala na određenim mjestima (priključne stanice) i tipično koriste neku vrstu provjere autentičnosti i praćenja korisnika (npr. primjenom elektronske pretplatničke kartice) kako bi mogući incidenti krađe bili izbjegnuti (Slika 1), ali sve brži razvoj tehnologija utro je put za razvoj nove generacije sustava dijeljenja poznatu pod nazivom *demand-responsive multimodal system*. Takvi sustavi biti će temeljeni na: fleksibilnim priključnim stanicama (premještanima prema intenzitetima pristupa sustavu na svakoj stanici i zahtjevima korisnika), poticanju smanjenja cijene korištenja ili dobivanju kredita za isporuku bicikala na prazne stanice, integracije *bike-sharinga* u javni prijevoz i mjestu *auto-sharinga* (putem pametnih kartica koje podržavaju brojne načine prijevoza) i na praćenju temeljenom na **GPS** (*engl. Global Positioning System*) tehnologiji. Moderne mrežne usluge kao što je *Social Bicycles* (SoBi), omogućuju korisnicima lociranje, rezervaciju, i otključavanje bicikala uz pomoć *smartphone* aplikacija, a također koriste sheme nagrađivanja kako bi motivirali korisnike da vrate bicikle na središnje stanice.

Slika 1: Prikaz Bixi Toronto sustava za dijeljenje bicikala [2]

Slično kao i kod sustava za dijeljenje bicikala, dijeljenje automobila predstavlja model kratkoročnog najma automobila, posebno atraktivan korisnicima koji zahtijevaju samo povremeno korištenje vozila, omogućujući pritom prednosti privatnih automobila bez ulaska u troškove i odgovornosti vlasništva. Dijeljenje automobila (*engl. car sharing*) prvi put se pojavilo u Sjevernoj Americi oko 1994. godine. Zamjena privatnih automobila s onim zajedničkim izravno smanjuje potražnju za parkirnim mjestima i prometne gužve u razdobljima najvećeg prometa, čime se podržava vizija održivog prijevoza. Operateri sustava za dijeljenje automobila obično omogućuju korisnicima preuzimanje automobila s određenih stanica (depoa), te njihovo vraćanje na izvorna mjesta nakon obavljene vožnje. Takvi programi nazivaju se dvosmjernim, odnosno *two-way* sustavima za dijeljenje automobila.

Većina do sada implementiranih realnih sustava nije bila sklona uvođenju inovativnih značajki (npr. jednosmjernih najmova, u kojima korisnik nije nužan vratiti vozilo na izvorno mjesto; *ridesharing*-a) zbog dodanih složenosti upravljanja. Upravo te složenosti bile su odgovorne za neuspjeh Hondinog *Diracc* sustava u Singapuru, jednog od najpoznatijih eksperimenata jednosmjernog sustava za dijeljenje automobila na svijetu, nakon 6 godina rada

(sustav je ukinut u 2008. godini). *Diracc* nije uspio uglavnom zato što nije bio sposoban zadržati kvalitete usluge (npr. dostupnost automobila) korisnicima jednosmjernog putovanja, odnosno zbog značajne neravnoteže sustava u zalihama vozila (tijekom uobičajenog dana, broj automobila u cijeloj mreži pomiče se prema pojedinim destinacijama; primjerice, vozači koji putuju iz predgrađa do centra grada stvaraju višak vozila na određenim postajama, dok iscrpljuju flote na ostalim postajama). Svejedno, neke nedavne inicijative dijeljenja automobila i dalje su vrlo uspješne, osobito Daimlerova *Car2Go* i BMW-ova *DriveNow* (Slika 2) koje nude mogućnost jednosmjernog dijeljenja automobila sve dok korisnik ne ostavi vozilo na bilo kojem dostupnom javnom parkiralištu, u određenom radnom okružju.

Slika 2: Prikaz BMW-ovog DriveNow sustava za dijeljenje električnih automobila [3]

Izvedba i upravljanje sustavom za dijeljenje automobila iziskuje rješavanje nekoliko optimizacijskih problema. Najprije treba odrediti optimalne veličine flota zajedno s mjestima parkirnih postaja. Nadalje, operateri koji pruže uslugu jednosmjerne vožnje trebaju razviti strategije za preraspodjelu vozila kako bi obnavljali optimalnu distribuciju flote među stanicama. Takav raspored može uzimati u obzir kratkoročne potrebe određenih stanica ili se

temeljiti na povijesnim predviđanjima (npr. procjeni buduće potražnje). Dok operateri sustava dijeljenja bicikala tipično koriste namjenska vozila za premještanje veće količine bicikala na stanice s iscrpljenim zalihama bicikala, premještanje vozila u programu dijeljenja automobila mnogo je zahtjevnije. Općenito, aktivnosti premještanja vozila mogu biti izvedene od strane korisnika ili operatera. U prvom slučaju, korisnika se potiče na odabir određene lokacije ili drugo vrijeme rezervacije; u drugom slučaju, koji je trenutno uobičajeni, vozila su fizički transportirana uz pomoć namjenskih kamiona ili specijalnog osoblja.

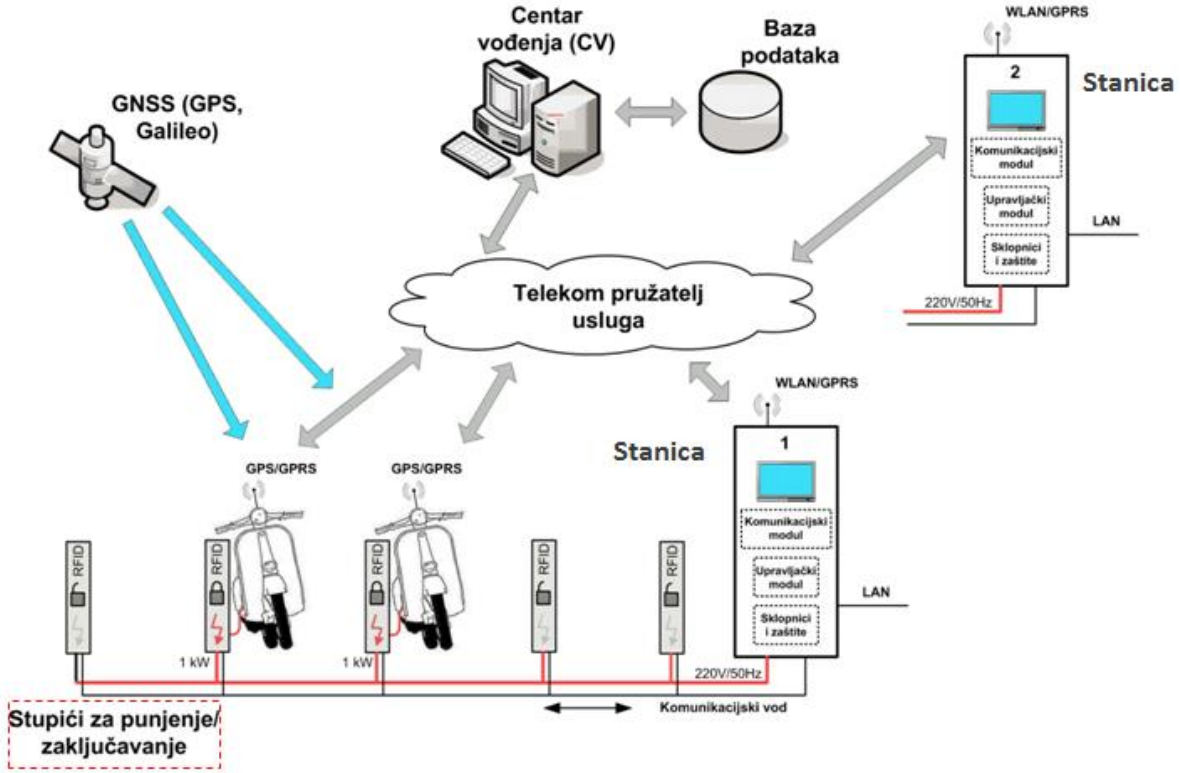
Najnoviji trendovi u sustavima za dijeljenje vozila usmjereni su na zamjenu konvencionalnih vozila s potpuno električnim vozilima (EV) s ciljem smanjenja negativnog utjecaja na okoliš u urbanim sredinama. Prilikom projektiranja sustava za dijeljenje EV-a potrebno je u obzir uzeti dostupnost stanica za punjenje na parkirnim mjestima i planiranje strategije premještanja koja uzima u obzir preostalu energiju vozila (na koji način i u kojem vremenskom razdoblju premještati vozila uzimajući u obzir stanja napunjenosti baterije vozila). Navedene izazove moguće je riješiti samo inteligentnim algoritamskim rješenjima s ciljem dugoročne održivosti sustava. Takvi algoritamski pristupi trebaju biti usmjereni na najveću moguću kvalitetu usluge za korisnike i smanjenje kapitalnih investicija za operatore. Za postizanje tih ciljeva, potrebno je riješiti cijeli spektar razvojnih i operativnih parametara svojstvenih sustavima za dijeljenje vozila kao što su: dugoročno strateško planiranje sustava i taktičke odluke vezane za dobrobit sustava te operativne probleme [1].

1.2. O sustavu za dijeljenje e-mopeda

Sustav za dijeljenje električnih mopeda jedan je od najmodernijih i najnaprednijih načina prijevoza čija je ideja proizašla iz poslovnog koncepta poznatog kao "ekonomija kolaborativne potrošnje", odnosno "ekonomija dijeljenja". Slično kao i kod dobro uspostavljenih usluga za dijeljenje automobila kao što je *Car2go*, dijeljenje e-mopeda će omogućiti ljudima pristup floti e-mopeda raspodijeljenih po čitavoj površini grada kako bi ih koristili kada god im je potrebno, a zatim odložili bilo gdje unutar određene gradske zone ili na posebnim stanicama za odlaganje, odnosno parkiranje e-mopeda.

Članovi sustava za dijeljenje e-mopeda usluge će plaćati prema korištenju, izbjegavajući na taj način sve skrivene troškove tipične kod posjedovanja vlastitog vozila kao što su: osiguranje, amortizacija, održavanje te nužna oprema kao što je kaciga, itd [4]. *Saturna Green Systems Scooter Sharing* predstavlja jednu od najuspješnijih implementacija sustava za

dijeljenje e-mopeda sa sjedištem u Vancouveru (Kanada), dok su u većini europskih gradova sustavi u probnoj fazi ili tek trebaju biti pokrenuti.



Slika 3: Struktura zatvorenog tipa sustava za dijeljenje električnih mopeda [5]

Postoje tri vrste sustava za dijeljenje električnih mopeda, svaka sa svojom zasebnom strukturom, a one su sljedeće [5]:

- ***Zatvoreni sustav dijeljenja:***

  Sastoji se od centra vođenja i stanica zajedno s njima pripadnim parkirnim mjestima te flote e-mopeda. Glavna karakteristika zatvorenog sustava dijeljenja jest ta da se na parkirnim mjestima pojedinih stanica prilikom parkiranja ili preuzimanja e-mopeda vrši njihovo mehaničko zabravljivanje ili odbravljivanje (Slika 3).

- ***Poluzatvoreni sustav dijeljenja***

  Gotovo identičan kao i zatvoreni sustav dijeljenja samo što se prilikom parkiranja ili preuzimanja e-mopeda ne vrši njihovo mehaničko zabravljivanje ili odbravljivanje.

- ***Otvoreni sustav dijeljenja***

  Sastoji se od centra vođenja i flote e-mopeda, bez stanica. Registracija korisnika i rezervacija e-mopeda vrši se preko mobilne aplikacije koja prikazuje najbliže e-mopede na digitalnoj karti i omogućuje odabir željenog e-mopeda jednostavnim

klikom na njega. E-mopedi se nakon uporabe ostavljaju nasumično ili na točno određenim lokacijama ovisno o zahtjevu vlasnika usluge (Slika 4).

Slika 4: Princip rada otvorenog tipa sustava za dijeljenje električnih mopeda [4]

U ovom radu projektirano je idejno rješenje sustava za dijeljenje električnih mopeda poluzatvorenog tipa za primjer grada Dubrovnika, koje se sastoji od detaljnog opisa principa rada sustava te prijedloga softverskog rješenja centra vođenja baziranog na računarstvu u oblacima (*engl. Cloud Computing*), odnosno na mrežnom (*engl. web*) serveru i njemu pripadnim klijentskim aplikacijama [6].

2. OPIS SUSTAVA

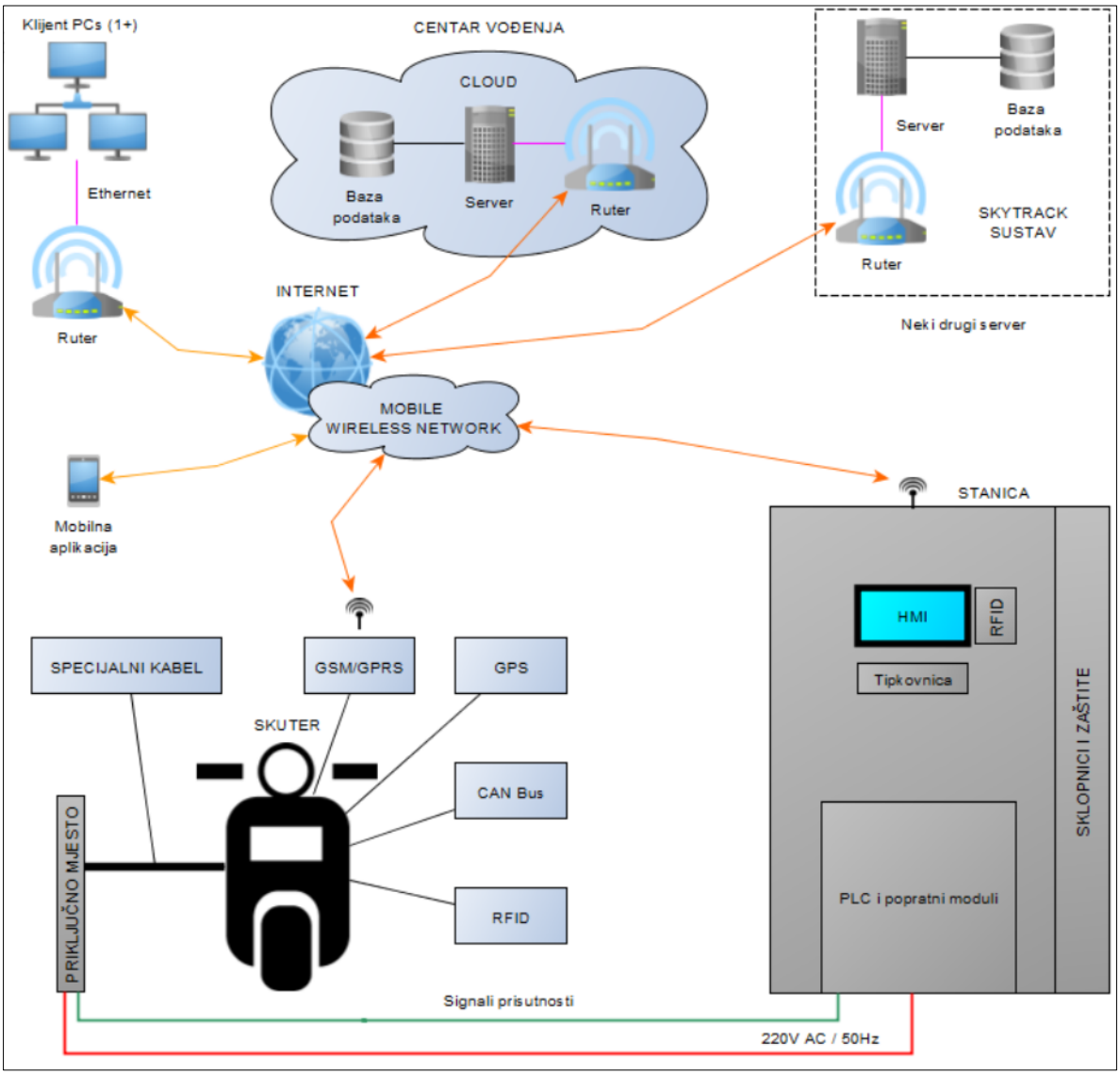
Sustav za dijeljenje električnih mopeda sastoji se od centra vođenja, određenog broja stanica te flote električnih mopeda. Centar vođenja je zadužen za potpuni nadzor sustava, odnosno praćenje stanja svih elemenata sustava i njihovo spremanje u bazu podataka, te za alarmiranje koje omogućava nadzornom osoblju brzo reagiranje na svaki izvanredni događaj. Pod praćenje stanja sustava spadaju sve radnje koje se odnose na zaprimanje podataka (komunikacijska razina) koje određeni elementi sustava (e-mopedi i stanice) šalju prema centru vođenja bilo vremenski periodički ili reakcijom na neki događaj, dok pod alarmiranje spadaju sve obavijesti ili upozorenja proizašla kao reakcija na zaprimljene podatke [6]. Neki od najbitnijih alarma su:

- e-moped izašao iz dozvoljenog područja
- stanje napunjenosti baterije e-mopeda manje od minimalnog dozvoljenog stanja, $SoC < SoC_{min}$
- prijavljen kvar ili nesreća tijekom korištenja e-mopeda

Stanica pak, za razliku od centra vođenja, djeluje samo na lokalnoj razini, odnosno upravlja samo podacima zaprimljenim od e-mopeda priključenih na njoj pripadna parkirna mjesta ili od korisnika putem integriranog **RFID** (*engl. Radio-frequency identification*) sustava. Također posjeduje grafička sučelja za ostvarivanje kontakta čovjek-stroj, odnosno **HMI**-a (*engl. Human machine interface*) putem kojih korisnik dobiva upute za radnje koje želi izvršiti nad e-mopedom, bilo to preuzimanje ili vraćanje e-mopeda. Spojena je s centrom vođenja uz pomoć **GPRS** (*engl. General Packet Radio Service*) komunikacije što joj omogućuje slanje i zaprimanje podataka, koje je nužno zbog dobivanja određenih povratnih informacija vezanih za korisnike i e-mopede kao što su: autorizacija korisnika, provjera dostupnih e-mopeda, itd. Dijeljenje podataka između e-mopeda i stanice ostvaruje se **GPRS** komunikacijom preko centra vođenja, a nastupa nakon fizičkog spajanja ili odspajanja e-mopeda s priključnog mjesta stanice.

Električni moped je najniži element sustava, odnosno osnovno sredstvo za prijevoz korisnika s početne na željenu destinaciju. Sadrži integrirani sustav **GPS** za globalno pozicioniranje i **GSM** (*engl. Global System for Mobile Communications*)/**GPRS** sustav za bežičnu komunikaciju s centrom vođenja, te **RFID** sustav za naknadnu autorizaciju korisničkih podataka prilikom preuzimanja e-mopeda. Spaja se na stanicu putem specijalnog kabla kojime

se istovremeno ostvaruje energetsko povezivanje i digitalna signalizacija njegove prisutnosti ili odsutnosti.



Slika 5: Struktura poluzatvorenog sustava za dijeljenje električnih mopeda

Kao što prikazuje Slika 5, glavni server centra vođenja nalazi se u oblaku, te može biti spojen s bilo kojim drugim sustavom (npr. *SkyTrack* sustavom), odnosno njegovom bazom podataka putem interneta s ciljem preuzimanja podataka. Klijentski se pak uređaji, bilo računala ili mobiteli spajaju preko određenih aplikacija na server centra vođenja putem interneta ili mobilne bežične mreže.

Aplikacija centra vođenja može se nalaziti, odnosno biti instalirana na jednom ili više klijentskih računala, a uz nju može biti izrađena i dodatna mrežna aplikacija čija je izvedba opisana u poglavlju 3. Kako bi princip rada sustava za dijeljenje električnih mopeda bio što bolje opisan sustav je raščlanjen na sljedeće procese (Tablica 1):

Tablica 1: Procesi pojedinih elemenata sustava

	CENTAR VOĐENJA	STANICA	E-MOPED
PROCESI	Zahtjevi i odgovori	Preuzimanje e-mopeda	Slanje podataka (periodičko)
	Upravljanje bazom podataka	Vraćanje e-mopeda	

2.1. Postupci preuzimanja i vraćanja e-mopeda

Preuzimanje e-mopeda na stanici vrši se prema sljedećim koracima [6]:

1. Prijava korisnika prislanjanjem kartice na **RFID** čitač koji se nalazi na stanici pored informativnog ekrana.
2. Slanje korisničkih podataka u centar vođenja **GPRS**-om radi njegove autorizacije i provjere statusa posjedovanja e-mopeda.
3. Obrada podataka u centru vođenja i slanje povratnih informacija.
4. Zaprimanje i interpretacija povratnih informacija od centra vođenja. Ukoliko je autorizacija uspješno prošla i postoji raspoloživi e-moped sustav nastavlja dalje, dok u suprotnom dolazi do prikaza poruke o grešci i obustavljanja procesa. Za slučaj da korisnik već posjeduje e-moped radnja je vraćanje (koraci su navedeni u odlomku ispod).
5. Odabir e-mopeda i njegova ponuda korisniku putem prikaza njegove oznake na informativnom ekranu. Sustav odabire e-moped s najboljim stanjem napunjenosti baterije uvažavajući i prioritete (e-mopedi parkirani na parkirnom mjestu bez priključaka za punjenje imaju veći prioritet od onih koji se pune).
6. Slanje korisničkih podataka preko centra vođenja **GPRS**-om u e-moped.
7. Prikaz parkirnog mjesta odabranog e-mopeda na informativnom ekranu.
8. Čekanje na fizičko odspajanje utikača od utičnice određeni vremenski period $T_{čekanja}$. Ukoliko je isteklo vrijeme čekanja obustavlja se proces, u suprotnom je e-moped uspješno preuzet.
9. Slanje podataka o uspješnom preuzimanju e-mopeda u centar vođenja.
10. Prikaz poruke o uspješnom preuzimanju e-mopeda na informativnom ekranu.
11. Ubacivanje kartice u **RFID** čitač e-mopeda.
12. Autorizacija korisnika na strani e-mopeda iz prethodno ubačenih korisničkih podataka.
13. Električno paljenje e-mopeda ukoliko je autorizacija bila uspješna.

Tablica 2 prikazuje tok podataka za sve korake preuzimanja e-mopeda u kojima je došlo do dijeljenja podataka.

Tablica 2: Formati poslanih i zaprimljenih podataka pri preuzimanju e-mopeda

KORAK	PODACI	POŠILJATELJ/PRIMATELJ
1.	UID korisnika: integer	korisnik/stanica
2.	UID korisnika: integer UID stanice: integer	stanica/centar vođenja
3.	Autorizacija: true/false Korisnik posjeduje e-moped: true/false Broj raspoloživih e-mopeda: integer Podaci odabranog e-mopeda: • UID: integer • Oznaka: string • Parkirno mjesto: integer	centar vođenja/stanica
6.	UID korisnika: integer	centar vođenja/e-moped
9.	UID korisnika: integer UID e-mopeda: integer UID stanice: integer Potvrda: string ('preuzet')	stanica/centar vođenja
11.	UID korisnika: integer	korisnik/e-moped

Vraćanje e-mopeda na stanici vrši se prema sljedećim koracima [6]:

1. Prijava korisnika prislanjanjem kartice na **RFID** čitač koji se nalazi na stanici pored informativnog ekrana.
2. Slanje korisničkih podataka u centar vođenja **GPRS**-om radi autorizacije i provjere posjedovanja e-mopeda.
3. Obrada podataka u centru vođenja i slanje povratnih informacija.
4. Zaprimanje i interpretacija povratnih informacija od centra vođenja. Ukoliko je autorizacija uspješno prošla sustav nastavlja dalje, dok u suprotnom dolazi do prikaza poruke o grešci i obustavljanja procesa. Za slučaj da korisnik ne posjeduje e-moped radnja je preuzimanje (koraci su navedeni u odlomku iznad).
5. Provjera da li postoji slobodno parkirno mjesto.
   POSTOJI: odvijaju se koraci 9. na dalje.
   NE POSTOJI: odvijaju se koraci 6., 7. i 8.
6. Prikaz sugestije ostavljanja e-mopeda na parkirnom mjestu bez priključaka na informativnom ekranu, uz moguć odabir DA ili NE.
7. Ukoliko je odabran odgovor NE obustavlja se proces vraćanja e-mopeda, dok se u suprotnom šalju podaci o uspješnom vraćanju e-mopeda u centar vođenja.

8. Prikaz poruke o uspješnom vraćanju e-mopeda na informativnom ekranu.
9. Prikaz slobodnog parkirnog mjesta kojeg je sustav odabrao na informativnom ekranu.
10. Odbravljivanje poklopca utičnice kako bi korisnik imao pristup utikaču.
11. Čekanje na fizičko spajanje utikača i utičnice određeni vremenski period $T_{čekanja}$. Ukoliko je isteklo vrijeme čekanja obustavlja se proces, u suprotnom slijedi zabravljivanje spoja utikača i utičnice.
12. Uključivanje sklopnika za punjenje e-mopeda.
13. Slanje podataka o uspješnom vraćanju e-mopeda u centar vođenja.
14. Prikaz poruke o uspješnom vraćanju e-mopeda na informativnom ekranu.

Tablica 3 prikazuje tok podataka za sve korake vraćanja e-mopeda u kojima je došlo do dijeljenja podataka.

Tablica 3: Formati poslanih i zaprimljenih podataka pri vraćanju e-mopeda

KORAK	PODACI	POŠILJATELJ/PRIMATELJ
1.	UID korisnika: integer	korisnik/stanica
2.	UID korisnika: integer UID stanice: integer	stanica/centar vođenja
3.	Autorizacija: true/false Korisnik posjeduje e-moped: true/false Broj slobodnih parkirnih mjesta: integer Odabrano parkirno mjesto: integer	centar vođenja/stanica
7.	UID korisnika: integer UID stanice: integer UID e-mopeda: integer Potvrda: string ('ostavljen')	stanica/centar vođenja
13.	UID korisnika: integer UID e-mopeda: integer UID stanice: integer Potvrda: string ('vraćen') Broj parkirnog mjesta: integer	stanica/centar vođenja

2.2. Zahtjevi i odgovori servera

Tijekom rada sustava za dijeljenje električnih mopeda centar vođenja je zadužen za obavljanje najznačajnijih radnji pri zaprimanju podataka od mnogo izvora (stanice, e-mopedi). Tablica 4 navodi sve radnje koje se odrađuju na strani servera centra vođenja, zajedno s njihovim ulaznim i izlaznim tipovima podataka te programskim kodom.

Pod autorizacijom korisnika podrazumijeva se provjera postojanja njegovog identifikacijskog broja (upisanog u **RFID** karticu) u bazi podataka nakon njegove prijave na stanicu. Jednom kada su dohvaćeni podaci korisnika (uz uspješnu autorizaciju) provjerava se status

posjedovanja e-mopeda čija se vrijednost ažurira nakon njegovog uspješnog preuzimanja, vraćanja ili ostavljanja na stanici.

Prilikom odabira e-mopeda tijekom odvijanja procesa njegova preuzimanja, sustav najprije provjerava da li za pripadnu stanicu postoje ostavljeni e-mopedi (viši prioritet naspram parkiranih), te ukoliko postoje odabire ostavljeni e-moped s najvećom vrijednosti stanja napunjenosti baterije (*SoC*), dok u suprotnom odabire parkirani e-moped prema istom principu. S druge strane, tijekom odvijanja procesa vraćanja e-mopeda, sustav odabire prvo slobodno parkirno mjesto u listi slobodnih parkirnih mjesta dohvaćene iz baze podataka. Pod ažuriranje podataka odvijenog procesa podrazumijevaju se sve izmjene u tablicama baze podataka.

Tablica 4: Funkcije centra vođenja zajedno s ulaznim i izlaznim tipovima podataka

<table>
<tr><td colspan="2">1. AUTORIZACIJA KORISNIKA</td></tr>
<tr><td>Ulazni podaci:</td><td>UID korisnika: integer</td></tr>
<tr><td colspan="2">Algoritam funkcije:</td></tr>
<tr><td colspan="2">

```
def autorizacijaKorisnika(uid_korisnika):
    podaci_korisnika = baza(baza.Korisnici.uid == uid_korisnika).select().first()
    if podaci_korisnika == None:
        return False
    if podaci_korisnika.zabranjen_pristup == True:
        return False
    else:
        return True
```

</td></tr>
<tr><td>Izlazni podaci:</td><td>Autorizacija uspješna: True | False</td></tr>
<tr><td colspan="2">2. PROVJERA POSJEDOVANJA E-MOPEDA</td></tr>
<tr><td>Ulazni podaci:</td><td>UID korisnika: integer</td></tr>
<tr><td colspan="2">Algoritam funkcije:</td></tr>
<tr><td colspan="2">

```
def provjeraPosjedovanjaSkutera(uid_korisnika):
    try:
        podaci_korisnika = baza(baza.Korisnici.uid == uid_korisnika)
                           .select().first()
        posjeduje_skuter = podaci_korisnika.posjeduje_skuter
        return bool(posjeduje_skuter)
    except:
        return None
```

</td></tr>
<tr><td>Izlazni podaci:</td><td>Posjeduje: True | False</td></tr>
<tr><td colspan="2">3. ODABIR E-MOPEDA</td></tr>
<tr><td>Ulazni podaci:</td><td>UID stanice: integer</td></tr>
<tr><td colspan="2">Algoritam funkcije:</td></tr>
<tr><td colspan="2">

```
def odabirSkuteraStanice(uid_stanice):
    podaci_stanice = baza(baza.Stanice.uid == uid_stanice).select().first()
    parkirani_skuteri = podaci_stanice.parkirani_skuteri
```

</td></tr>
</table>

```
    ostavljeni_skuteri = podaci_stanice.ostavljeni_skuteri
    len_park = len(parkirani_skuteri)
    len_ost = len(ostavljeni_skuteri)
    broj_skutera = len_park + len_ost
    if broj_skutera > 0:
        SoC_najveci = 0.0
        odabrani_skuter = None
        if len_ost > 0:
            for skuter_uid in ostavljeni_skuteri:
                SoC_skutera = baza(baza.Skuteri.uid == skuter_uid)
                              .select().first().SoC
                if SoC_skutera > SoC_najveci:
                    SoC_najveci = SoC_skutera
                    odabrani_skuter = skuter_uid
        else:
            for skuter_uid in parkirani_skuteri:
                SoC_skutera = baza(baza.Skuteri.uid == skuter_uid)
                              .select().first().SoC
                if SoC_skutera > SoC_najveci:
                    SoC_najveci = SoC_skutera
                    odabrani_skuter = skuter_uid
        podaci_skutera = baza(baza.Skuteri.uid == odabrani_skuter)
                         .select().first()
        rez = {'uid':podaci_skutera.uid,
               'parkirno_mjesto':podaci_skutera.parkirno_mjesto,
               'oznaka':podaci_skutera.oznaka}
    else: rez = {'uid':None, 'parkirno_mjesto':None, 'oznaka':None}
    return rez
```

Izlazni podaci:	Broj raspoloživih e-mopeda: integer Podaci odabranog e-mopeda: • UID: integer • Oznaka: string • Broj parkirnog mjesta: integer
4. ODABIR PARKIRNOG MJESTA	
Ulazni podaci:	UID stanice: integer
Algoritam funkcije:	

```
def odabirParkirnogMjesta(uid_stanice):
    podaci_stanice = baza(baza.Stanice.uid == uid_stanice).select().first()
    spm = podaci_stanice.slobodna_parkirna_mjesta
    br = len(spm)
    if br > 0:
        pm = spm.pop()
        rez = {'parkirno_mjesto':pm, 'broj_slobodnih_parkirnih_mjesta':br}
    else:
        rez = {'parkirno_mjesto':None, 'broj_slobodnih_parkirnih_mjesta':None}
    return rez
```

Izlazni podaci:	Broj slobodnih parkirnih mjesta: integer Broj odabranog parkirnog mjesta: integer
5. AŽURIRANJE USPJEŠNOG PREUZIMANJA/VRAĆANJA/OSTAVLJANJA E-MOPEDA	
Ulazni podaci:	UID korisnika: integer UID e-mopeda: integer UID stanice: integer

	Parkirno mjesto: integer Status: string ("preuzet | vraćen | ostavljen")

Algoritam funkcije:

```
def azuriranjeIzvrseneRadnje(uid_korisnika, uid_stanice, uid_skutera, parkirno_mj,
                             status, vrijeme):
    pm = baza(baza.Skuteri.uid == uid_skutera).select().first().parkirno_mjesto
    podaci_stanice = baza(baza.Stanice.uid == uid_stanice).select().first()
    if status == 'preuzet':
        baza(baza.Korisnici.uid == uid_korisnika).update(posjeduje_skuter=True)
        baza(baza.Skuteri.uid == uid_skutera).update(status='aktivan',
             punjenje=False, stanica=0, korisnik=uid_korisnika, parkirno_mjesto=0)
        if pm > 0:
            ps = podaci_stanice.parkirani_skuteri
            spm = podaci_stanice.slobodna_parkirna_mjesta
            ps.remove(uid_skutera)
            spm.append(parkirno_mj)
            baza(baza.Stanice.uid == uid_stanice).update(parkirani_skuteri=ps,
                                                         slobodna_parkirna_mjesta=spm)
        else:
            os = podaci_stanice.ostavljeni_skuteri
            os.remove(uid_skutera)
            baza(baza.Stanice.uid == uid_stanice).update(ostavljeni_skuteri=os)
    elif status == 'vracen':
        baza(baza.Korisnici.uid == uid_korisnika).update(posjeduje_skuter=False)
        baza(baza.Skuteri.uid == uid_skutera).update(status='parkiran',
             punjenje=True, stanica=uid_stanice, korisnik=0,
             parkirno_mjesto=parkirno_mj)
        ps = podaci_stanice.parkirani_skuteri
        ps.append(uid_skutera)
        spm = podaci_stanice.slobodna_parkirna_mjesta
        spm.remove(parkirno_mj)
        baza(baza.Stanice.uid == uid_stanice).update(parkirani_skuteri=ps,
                                                     slobodna_parkirna_mjesta=spm)
    elif status == 'ostavljen':
        baza(baza.Korisnici.uid == uid_korisnika).update(posjeduje_skuter=False)
        baza(baza.Skuteri.uid == uid_skutera).update(status='ostavljen',
             punjenje=False, stanica=uid_stanice, korisnik=0, parkirno_mjesto=0)
        os = podaci_stanice.ostavljeni_skuteri
        os.append(uid_skutera)
        baza(baza.Stanice.uid == uid_stanice).update(ostavljeni_skuteri=os)
    baza.Pristupi_sustavu.insert(**{'uid_skutera':uid_skutera,
                                    'uid_stanice':uid_stanice,
                                    'uid_korisnika':uid_korisnika,
                                    'radnja':status,
                                    'vrijeme':sekunde2vrijeme(vrijeme)})
    return 'Skuter %s.' % status
```

Izlazni podaci:	Nema izlaznih podataka (vrše se samo izmjene u tablicama baze podataka)

2.3. Upravljanje bazom podataka

Upravljanje podacima iz baze podataka vrši se na serverskoj strani, odnosno na strani centra vođenja. Kako bi se podacima uspješno rukovalo potrebno je najprije ostvariti povezivanje s bazom podataka, te nakon toga programskim upitima (*engl. query*) izvršiti željene operacije

(npr. čitanje ili pisanje) nad pripadnim poljima određenih tablica. Povezivanje s bazom podataka ostvaruje se na sljedeći način uz pomoć *Python*-ovog *sqlite3* modula [7]:

```
import sqlite3 as sql

db = sql.connect("./Baze/EMS_init.sqlite")
selektor = db.cursor()
```

Nakon toga, potrebno je izvršiti naredbu koja kreira tablice ukoliko već ne postoje:

```
db.execute('''
CREATE TABLE IF NOT EXISTS Stanice(
    uid INTEGER PRIMARY KEY,
    lokacija TEXT,
    napajanje INTEGER,
    punjenje INTEGER,
    komunikacija INTEGER,
    lat DOUBLE,
    lon DOUBLE,
    broj_skutera_inicijalni INTEGER,
    broj_parkirnih_mjesta INTEGER,
    parkirani_skuteri TEXT,
    ostavljeni_skuteri TEXT,
    slobodna_parkirna_mjesta TEXT,
    tezina DOUBLE,
    centar INTEGER
);''')
db.commit()
```

Jednom kada su tablice definirane, naredbom INSERT željeni podaci ubacuju se u njihova polja, a naredbom UPDATE ažuriraju se vrijednosti pripadnih polja [8]. Tablice baze podataka sustava za dijeljenje električnih mopeda definirane su kako slijedi [6]:

- Korisnici - sadrži sve podatke vezane za pojedinog korisnika (Tablica 5).

Tablica 5: Definirana polja vezana za korisnike

IME POLJA	TIP POLJA
UID	INTEGER PRIMARY KEY
Ime	CHAR(512)
Prezime	CHAR(512)
Korisničko ime	CHAR(512) UNIQUE
Email adresa	CHAR(512) UNIQUE
Šifra	CHAR(512)
Tarifa	CHAR(512)
Vrijedi od	TIMESTAMP
Vrijedi do	TIMESTAMP
Zabranjen pristup	INTEGER
Posjeduje skuter	INTEGER

- Stanice - sadrži sve podatke vezane za pojedine stanice (Tablica 6).

Tablica 6: Definirana polja vezana za stanice

IME POLJA	TIP POLJA
UID	INTEGER PRIMARY KEY

Lokacija	TEXT
Napajanje	INTEGER
Punjenje	INTEGER
Komunikacija	INTEGER
Geografska širina	DOUBLE
Geografska dužina	DOUBLE
Broj skutera (inicijalni)	INTEGER
Broj parkirnih mjesta	INTEGER
Parkirani skuteri	TEXT
Ostavljeni skuteri	TEXT
Slobodna parkirna mjesta	TEXT
Težina	DOUBLE
Centar	INTEGER

- E-mopedi - sadrži sve podatke vezane za pojedine e-mopede (Tablica 7).

Tablica 7: Definirana polja vezana za e-mopede

IME POLJA	TIP POLJA
UID	INTEGER PRIMARY KEY
Tip	CHAR(512)
Oznaka	CHAR(512)
Status	CHAR(512)
SoC	DOUBLE
SoH	DOUBLE
Geografska širina	DOUBLE
Geografska dužina	DOUBLE
Punjenje	INTEGER
Parkirno mjesto	INTEGER
Stanica	INTEGER
Korisnik	INTEGER
Broj ciklusa punjenja	INTEGER
Datum zadnjeg punjenja	TIMESTAMP
Ukupni put	DOUBLE

- Pristupi sustavu - sadrži sve podatke koji se spremaju reakcijom na neki događaj, odnosno koji su poslani prema centru vođenja prilikom uspješnog preuzimanja, vraćanja ili ostavljanja e-mopeda na parkirnom mjestu bez priključaka. Uz pomoć ove tablice nadzornom osoblju se omogućuje uvid u povijest svih radnji ili filtracija podataka prema jednom ili više od njoj pripadnih polja (Tablica 8).

Tablica 8: Definirana polja vezana za pristupe sustavu

IME POLJA	TIP POLJA
UID korisnika	INTEGER REFERENCES Korisnici (uid) ON DELETE CASCADE
UID stanice	INTEGER REFERENCES Stanice (uid) ON DELETE CASCADE
UID skutera	INTEGER REFERENCES Skuteri (uid) ON DELETE CASCADE
Vrijeme	TIMESTAMP
Radnja	CHAR(512)

- Stanja e-mopeda - sadrži sve podatke koje e-moped periodički šalje prema centru vođenja. Uz pomoć ove tablice nadzornom osoblju je omogućena potpuna kontrola i uvid u stanja svih e-mopeda kroz čitavo vrijeme rada sustava (Tablica 9).

Tablica 9: Definirana polja vezana za stanja e-mopeda

IME POLJA	TIP POLJA
UID e-mopeda	INTEGER
SoC	DOUBLE
SoH	DOUBLE
Geografska širina	DOUBLE
Geografska dužina	DOUBLE
Temperatura	DOUBLE
Napon	DOUBLE
Struja	DOUBLE
Status	CHAR(512)
Punjenje	INTEGER
Stanica	INTEGER
Korisnik	INTEGER
Greške	TEXT
Vrijeme	TIMESTAMP

2.4. Periodičko slanje podataka

Svi e-mopedi u sustavu šalju podatke u centar vođenja periodički (uz period T_{slanja}) kako bi se omogućio potpuni nadzor sustava te njihovo bilježenje (*engl. data logging*), odnosno spremanje u bazu podataka za potrebe daljnje obrade ili analize. Podaci koje e-moped periodički šalje uključuju: vlastiti UID (*engl. Unique Identifier*), stanje napunjenosti baterije (*SoC*), stanje zdravlja baterije (*SoH*), geografska širinu i dužinu (**GPS**), temperaturu, napon, struju, status (aktivan, parkiran, ostavljen), punjenje (aktivno/neaktivno), stanicu (UID stanice - nula ako nije priključen na stanicu), korisnika (UID korisnika - nula ako je parkiran ili ostavljen), greške, vrijeme slanja (u formatu *godina-mjesec-dan sat:minuta:sekunda*), te ukupni prijeđeni put.

Također, osim podataka e-mopeda šalju se i podaci vezani uz stanice, odnosno podaci o promjeni statusa punjenja ukoliko na nju nije priključen niti jedan e-moped ili je stanje napunjenosti baterije svakog e-mopeda jednako 100 %. Algoritam funkcije za detekciju promjene statusa punjenja stanice i njezin *Python* kod definirani su kako slijedi:

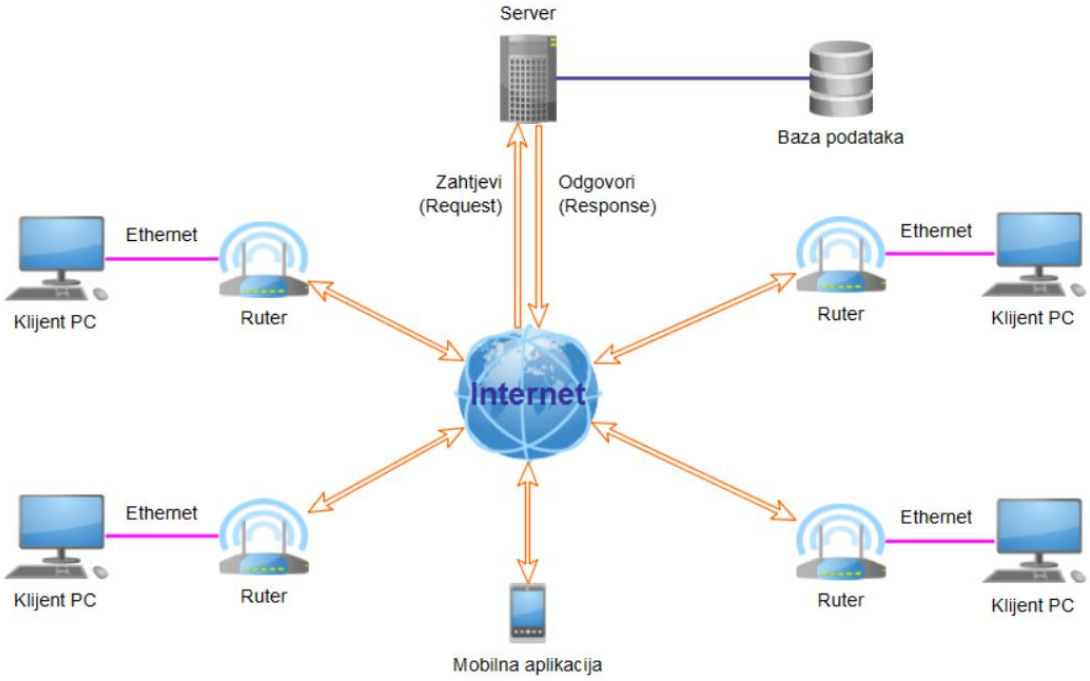
1. Učitaj trenutni status punjenja stanice
2. Dohvati listu trenutno parkiranih e-mopeda
3. Provjeri broj parkiranih e-mopeda i njihova stanja napunjenosti baterije (*SoC*)

4. Ukoliko ne postoje parkirani e-mopedi ili su svi puni, postavi novi status punjenja stanice na "isključeno", u suprotnom na "uključeno"
5. Ukoliko je novo stanje punjenja stanice različito od prethodnog stanja ažuriraj promjenu u centru vođenja

```
def posaljiPodatkeStanica(self):
    for stanica in self.lista_stanica:
        # Trenutni status punjenja
        punjenje = stanica.punjenje
        # Lista parkiranih e-mopeda
        ps = [self.dohvatiElementPoUID(uid, self.lista_skutera) for uid\
              in stanica.parkirani_skuteri]
        # Provjera broja parkiranih e-mopeda i njihovog SoC-a
        ps_bool = len(ps) == 0
        soc_bool = len([s for s in ps if s.soc < 100]) == 0
        # Ukoliko ne postoje parkirani e-mopedi ili su svi puni postavi
        # status punjenja na uključeno, u suprotnom na isključeno
        if ps_bool or soc_bool:
            stanica.punjenje = False
        else:
            stanica.punjenje = True
        # Ukoliko je novo stanje punjenja različito od prethodnog stanja
        # ažuriraj promjenu u centru vođenja
        if punjenje != stanica.punjenje:
            funkcija = "osvjeziStanjeStanice"
            argumenti = "?uid=%s&punjenje=%s" % (stanica.uid,
                                                 int(stanica.punjenje))
            odgovor = self.posaljiPodatkeCV(funkcija, argumenti, "POST")
```

3. IZVEDBA CENTRA VOĐENJA

Sustav centra vođenja sastoji se od servera, čija je glavna zadaća zaprimati zahtjeve od strane e-mopeda, stanica i klijentskih aplikacija, te nakon obrade ulaznih podataka slati određene odgovore. Server je u ovom slučaju realiziran kao **HTTP** (*engl. HyperText Transfer Protocol*) server koji se nalazi u oblaku pod vlasništvom *PythonAnywhere* pružatelja "web hostinga", odnosno mrežnog **IDE**-a (*engl. Integrated Development Environment*) baziranog na objektno orijentiranom programskom jeziku *Python* [9]. Svi upiti prema serveru u formi su **HTTP** upita (engl. *request*) tipa GET ili POST. GET naredbom zaprimaju se podaci sa servera, dok se POST naredbom podaci šalju prema serveru [10].



Slika 6: Prikaz odnosa servera i klijenata

PythonAnywhere usluga je zapravo model usluge računarstva u oblacima (*engl. Cloud Computing*) poznat kao **PaaS** (*engl. Platform as a Service*) kojeg karakterizira sljedeće [11]:

- uključuje pružanje usluge operacijskog sučelja za određenu aplikaciju
- korisnik pritom kontrolira aplikacije koje se tako pokreću, a po potrebi mu je pružena i kontrola nad samom uslugom hostinga
- korisnik nema kontrolu nad operacijskim sustavom, sklopovljem ili mrežnom infrastrukturom na kojoj se aplikacija pokreće

Slika 7: Struktura javnog oblaka [11]

Za potrebe simuliranja rada centra vođenja izrađena je mrežna aplikacija centra vođenja koja uključuje sve elemente sustava za dijeljenje električnih mopeda, odnosno pregled svih stanica, e-mopeda i korisnika, njihovo ubacivanje u bazu putem određenih obrazaca/formi, te glavne funkcije za zaprimanje zahtjeva i slanje odgovora. Aplikacija se nalazi na besplatnoj privatnoj domeni čija je adresa, odnosno **URL** (*engl. Uniform Resource Locator*): https://jt195996.pythonanywhere.com/EMS/default/index

3.1. Web2py razvojno okruženje

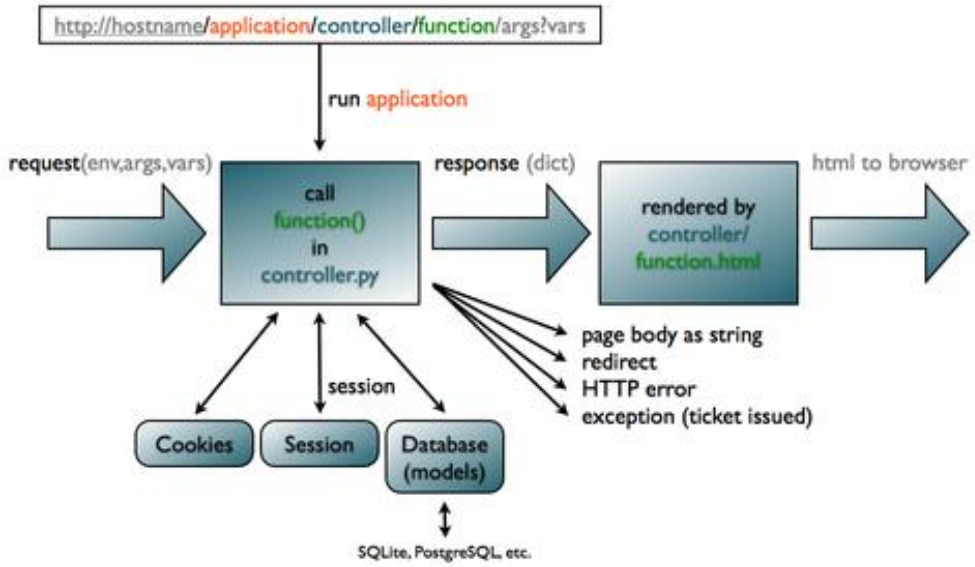
Web2py je besplatan program za izradu mrežnih aplikacija koji je napisan u *Python*-u i koji koristi *Python* za programiranje aplikacija. Za njegovo stvaranje najzaslužniji je Massimo Di Pierro. Jednostavan je za korištenje i ne zahtijeva nikakvu instalaciju i dodatnu konfiguraciju, a 2012. godine proglašen je također i tehnologijom godine. Unutar njemu pripadnog administratorskog sučelja moguće je razvijati nove aplikacije, obrisati neželjene aplikacije, te instalirati već postojeće aplikacije. Administratorsko sučelje također pruža uvid u sve mape, datoteke, baze podataka, module i funkcije svake aplikacije [12]. Web2py mrežni okvir moguće je preuzeti u obliku izvornog koda ili kao izvršnu (*engl. executable*) datoteku na njihovoj službenoj stranici http://www.web2py.com/init/default/download.

Svaka Web2py aplikacija sastoji se od modela (*engl. model*), pogleda (*engl. view*) i kontrolera (*engl. controller*) koji čine tzv. **MVC** (*engl. model-view-controller*) strukturu, te vremenskog raspoređivača poslova (*engl. Cron jobs*) za zadatke koji se moraju redovito izvršiti u pozadini u određenim vremenskim trenutcima, modula i statičkih datoteka kao što su slike, **CSS** (*engl.*

Cascading Style Sheet) skripte, *JavaScript* skripte, itd. U modelu se tipično nalaze definicije baza podataka i funkcija za koje želimo da nam budu uvijek dostupne, odnosno svaka definicija varijable i funkcije unutar modela poprima globalni doseg (*engl. scope*) bez da je se prethodno mora definirati globalnom. Web2py posjeduje integriranu **DAL** (*engl. Database Abstraction Layer*) klasu koja dinamički generira **SQL** (*engl. Structured Query Language*) sintaksu u realnom vremenu s pomoću određenih pravila za razne tipove baza podataka, omogučujući pritom programeru da ne mora poznavati sintakse različitih tipova baza podataka. Neke od podržanih tipova baza podataka su: SQLite, MySQL, Oracle, MSSQL, PostgreSQL, DB2, FireBird, itd.

Unutar kontrolera nalaze se funkcije čije ime mora biti vezano za pojedinu stranicu aplikacije, odnosno čije će povratne varijable biti dostupne samo unutar pregleda pripadne stranice. Tako npr. za stranicu koja se naziva *index.html* mora biti definirana funkcija čiji naziv odgovara nazivu te stranice (*index*) i koja ne posjeduje ulazne argumente. U njemu se također definiraju i funkcije koje se iz pregleda pozivaju uz pomoć **AJAX** (*engl. Asynchronous JavaScript and XML*) naredbi. Takve funkcije nisu vezane niti za jednu stranicu već se vežu za **HTML** element s određenim *id*-om, te se asinkrono izvršavaju unutar te stranice.

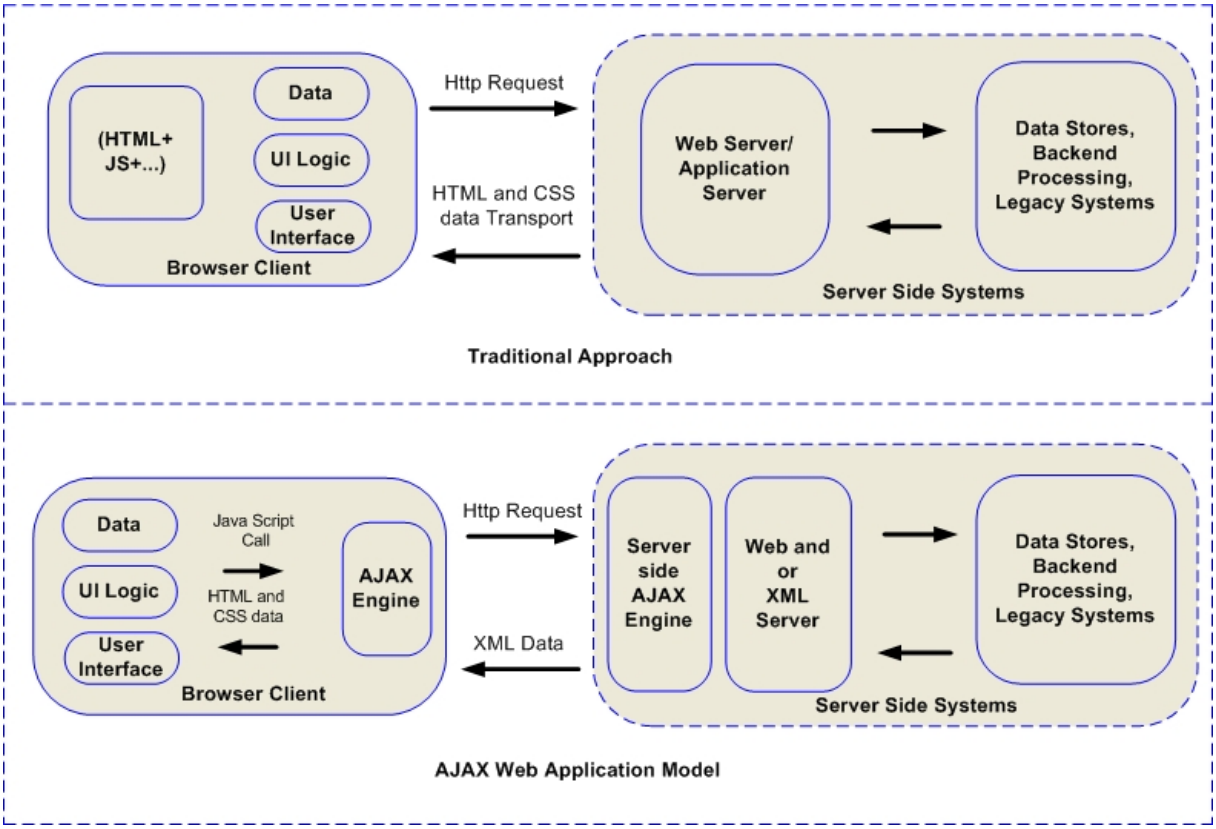
U pregledu se pak nalazi sav **HTML** kod nužan za definiranje strukture stranice, **CSS** kod nužan za definiranje stila stranice, te *JavaScript* kod koji korisniku pruža mogućnost interakcije i manipulacije nad elementima stranice unutar preglednika (*engl. browser*), ali ne i podacima na strani servera. Unutar njega je također moguće pisati i *Python* kod, ali onda moraju biti zadovoljena određena pravila.



Slika 8: Princip rada Web2py mrežnog okvira (engl. web framework) [13]

3.2. Opis mrežne aplikacije

Aplikacija centra vođenja sastoji se od pregleda stanica, e-mopeda, korisnika, zapisa servera i alarma, te od upravljanja bazom podataka. Svi odabrani podaci koji se prikazuju prethodno se dohvaćaju na strani servera iz određenih tablica baze podataka. Nakon inicijalizacije određene stranice programskim se upitima, odnosno funkcijama koje koriste **AJAX** tehnologiju, svakih $T_{ajax} = 5$ s ažuriraju njoj pripadni podaci. **AJAX** tehnologija omogućuje mrežnim aplikacijama slanje i dohvaćanje podataka sa servera asinkrono (u pozadini), odnosno bez ometanja ponašanja postojeće stranice.



Slika 9: Princip rada tradicionalne i AJAX mrežne aplikacije [14]

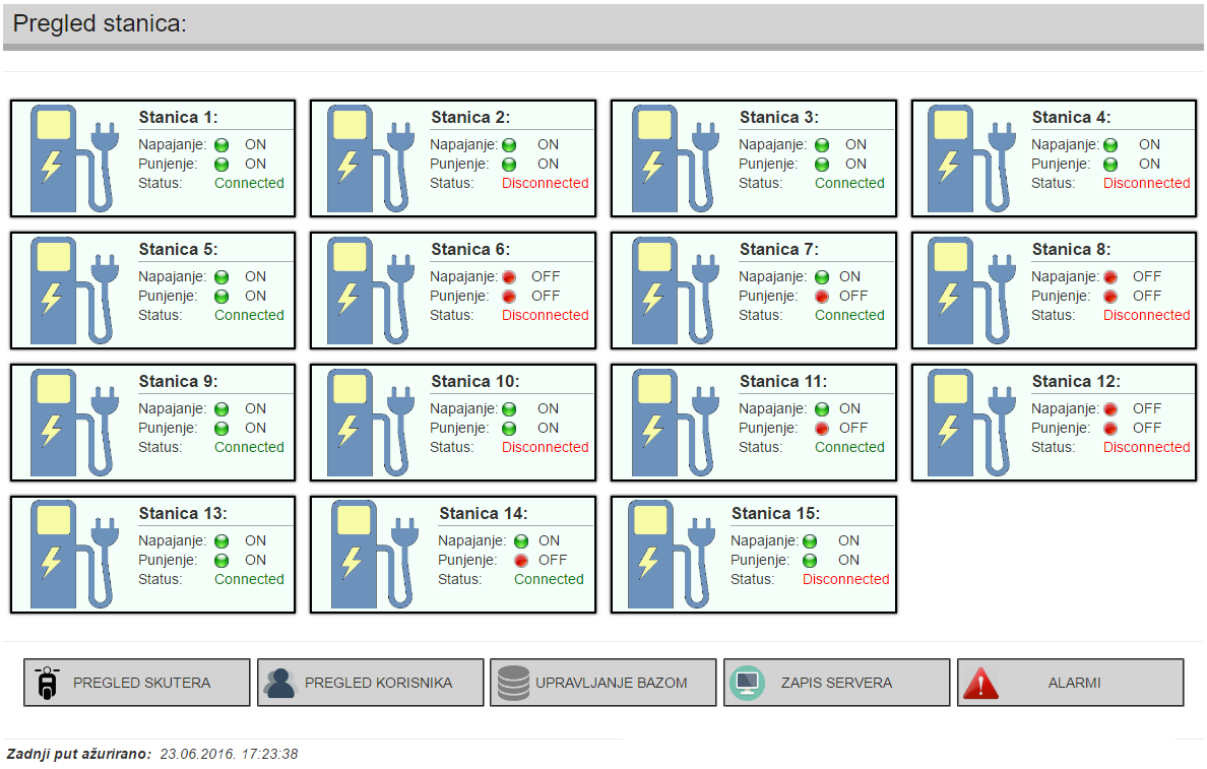
Struktura **AJAX** funkcije definirana je kako slijedi, a njeni glavni ulazni argumenti su **URL** funkcije koju želimo izvršiti, te dodatne postavke kao što je tip upita (GET ili POST), asinkronost zahtjeva, vrijeme isteka u milisekundama, itd. Funkcija također uključuje i posebno definirane događaje (*beforeSend, error, success i complete*) nakon kojih je moguće izvršiti vlastite blokove *JavaScript* koda [15].

```
$.ajax({
    url: "URL željene funkcije",
    type: 'get',
    async: true,
    timeout: 10000,
    data: {'uid':UID},
```

```
    beforeSend: function(){
        // kod koji se izvršava prije slanja podataka
    },
    error: function(xhr, status, error){
        // kod koji se izvršava ukoliko je došlo do greške
    },
    success: function(result){
        // kod koji se izvršava prilikom uspješnog izvršenja tražene
           funkcije
    },
    complete: function(){
        // kod koji se izvršava nakon izvršenja tražene funkcije neovisno o
           uspjehu
    }
});
```

3.2.1. Prikaz stanja stanica

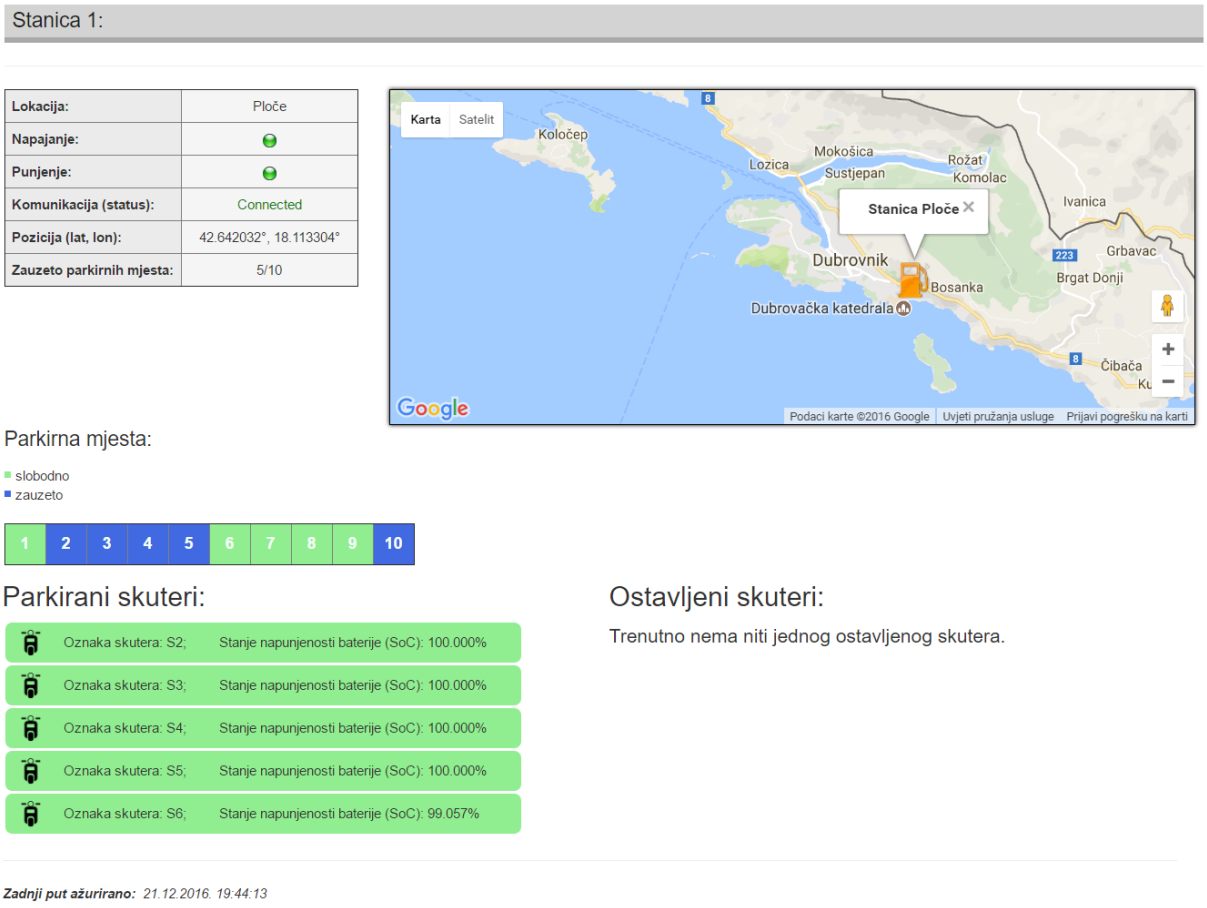
Na ekranu pregleda stanica grafički su prikazane sve postojeće stanice zajedno s pripadnim osnovnim podacima kao što su statusi napajanja, punjenja i komunikacije s centrom vođenja (Slika 10). Pritiskom na željenu stanicu učitava se nova stranica koja sadrži detaljan pregled svih podataka stanice (Slika 11).



Slika 10: Prikaz ekrana pregleda stanica aplikacije centra vođenja

Detaljan pregled stanice sadrži sve njene podatke kao što su: naziv lokacije, pozicija (**GPS** koordinate), broj zauzetih parkirnih mjesta, ukupan broj parkirnih mjesta, grafički prikaz trenutno zauzetih i slobodnih parkirnih mjesta, grafički prikaz stanice na *Google* mapi, lista svih parkiranih i ostavljenih e-mopeda na stanici zajedno s osnovnim im podacima i

mogućnošću daljnjeg prikaza detaljnih podataka pritiskom na željenu oznaku e-mopeda (vidi sliku Slika 11).



Slika 11: Prikaz pregleda podataka učitane stanice aplikacije centra vođenja

Određeni segmenti grafičkog sučelja koji koriste podatke čije se vrijednosti mijenjaju tijekom simulacije sustava vremenski se osvježavaju kako bi nadzorni tim konstantno imao uvid u trenutno stanje sustava.

3.2.2. Vremensko praćenje pozicija e-mopeda

Pritiskom na gumb "Pregled skutera" otvara se stranica koja sadrži *Google* mapu na kojoj se vremenski osvježavaju podaci pozicija svih e-mopeda, te se potom grafički iscrtavaju korištenjem *Google Maps* **API**-a (*engl. Application programming interface*), odnosno skupa metoda i alata koji se koriste za izgradnju softverskih aplikacija [16]. Prilikom inicijalizacije stranice također se grafički iscrtavaju i sve postojeće stanice.

Za uspješno korištenje *Google Map* **API**-a nužno je najprije dodati njenu *JavaScript* knjižicu (*engl. library*) u mrežnu stranicu:

```
<script src="https://maps.googleapis.com/maps/api/js?key=MOJ_KLJUC">
</script>
```

Zatim treba kreirati **HTML** element u kojem će se mapa nalaziti:

```
<div id="mapa" style="width:100%;height:100%"></div>
```

Nakon toga potrebno je definirati funkciju koja će prilikom učitanja dokumenta, odnosno mrežne stranice, pozvati funkciju za inicijalizaciju mape i pokrenuti ponavljajući vremenski brojač koji će pri svakom isteku vremena pokretati funkciju za osvježavanje grafike:

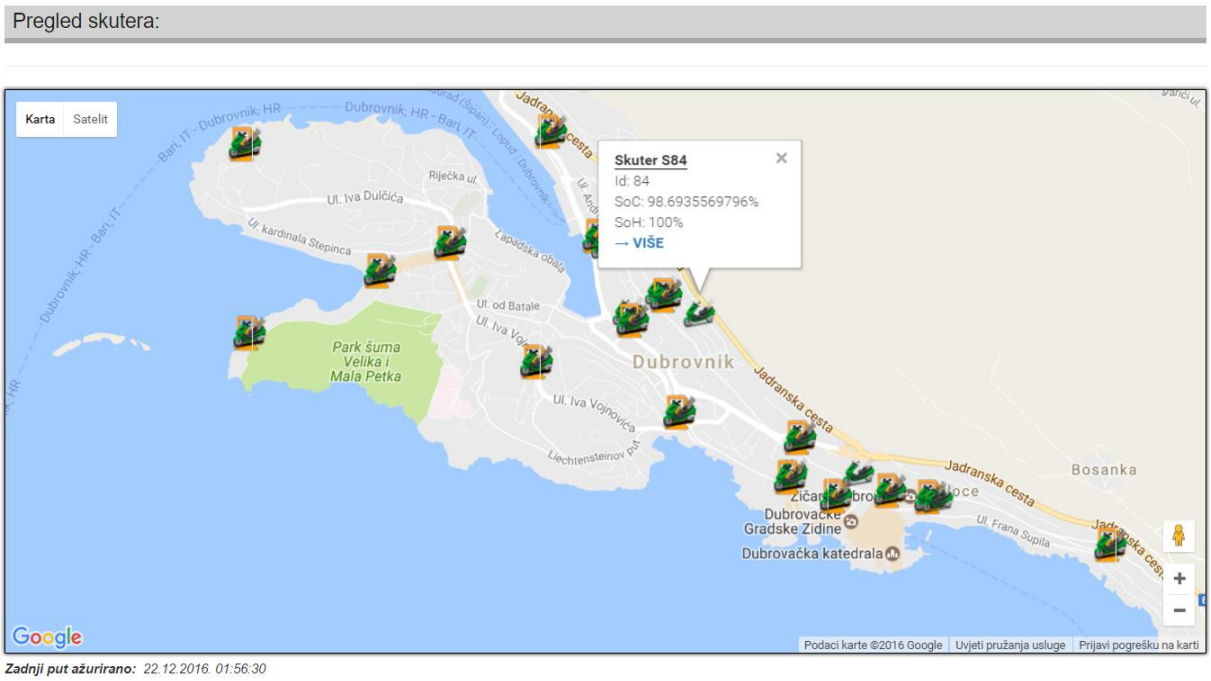
```
<script>
$(document).ready(function(){
    google.maps.event.addDomListener(window, 'load', inicijalizacijaMape);
    setInterval(function(){
        osvjeziGrafiku();
    }, 5000);
});
</script>
```

Mapa se inicijalizira tako da se najprije odredi centar (u ovom slučaju geografska širina i dužina centra Dubrovnika), zatim se definiraju određene postavke (razina zumiranja, tip mape, ...), te se potom kreira objekt mape (*new google.maps.Map()*). Funkcija za inicijalizaciju mape definirana je kako slijedi, a sadrži i inicijalizaciju objekata markera (*new google.maps.Marker()*) i info-prozora (*new google.maps.InfoWindow()*) za svaki e-moped i stanicu sustava:

```
<script>
function inicijalizacijaMape() {
    inicijalizirano = false;
    marker = [];
    prozor = [];
    marker_stanica = [];
    prozor_stanica = [];
    var n = {{=len(skuteri)}};
    var m = {{=len(stanice)}};
    var centar = new google.maps.LatLng(42.6501, 18.09444);
    var mapProp = {
        center: centar,
        zoom: 13,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    map = new google.maps.Map(document.getElementById("mapa"), mapProp);
    var i;
    for (i=0;i < n;i++){
        marker[i] = new google.maps.Marker({
            id: i,
            icon: "{{=URL('static', 'images/skuter.png')}}"
        });
        marker[i].setMap(map);
        prozor[i]= new google.maps.InfoWindow();
        google.maps.event.addListener(marker[i], 'click', function(event) {
            prozor[this.id].open(map,marker[this.id]);
        });
    }
    var j;
    for (j=0;j < m;j++){
```

```
        marker_stanica[j] = new google.maps.Marker({
            id: j,
            icon: "{{=URL('static', 'images/stanica.png')}}"
        });
        marker_stanica[j].setMap(map);
        prozor_stanica[j]= new google.maps.InfoWindow();
        google.maps.event.addListener(marker_stanica[j], 'click',
        function(event) {
            prozor_stanica[this.id].open(map,marker_stanica[this.id]);
        });
    }
}
</script>
```

Jednom inicijalizirana mapa s pripadnim joj objektima koristi se u funkciji za osvježavanje grafike kako bi se osvježile pozicije markera svih e-mopeda i kako bi se prilikom pritiska mišem na njih otvorio prozor koji sadrži njihove osnovne podatke kao što su: oznaka e-mopeda, UID e-mopeda, stanje napunjenosti baterije (*SoC*) i stanje zdravlja baterije (*SOH*), te poveznicu koja vodi do detaljnog prikaza svih podataka određenog e-mopeda.

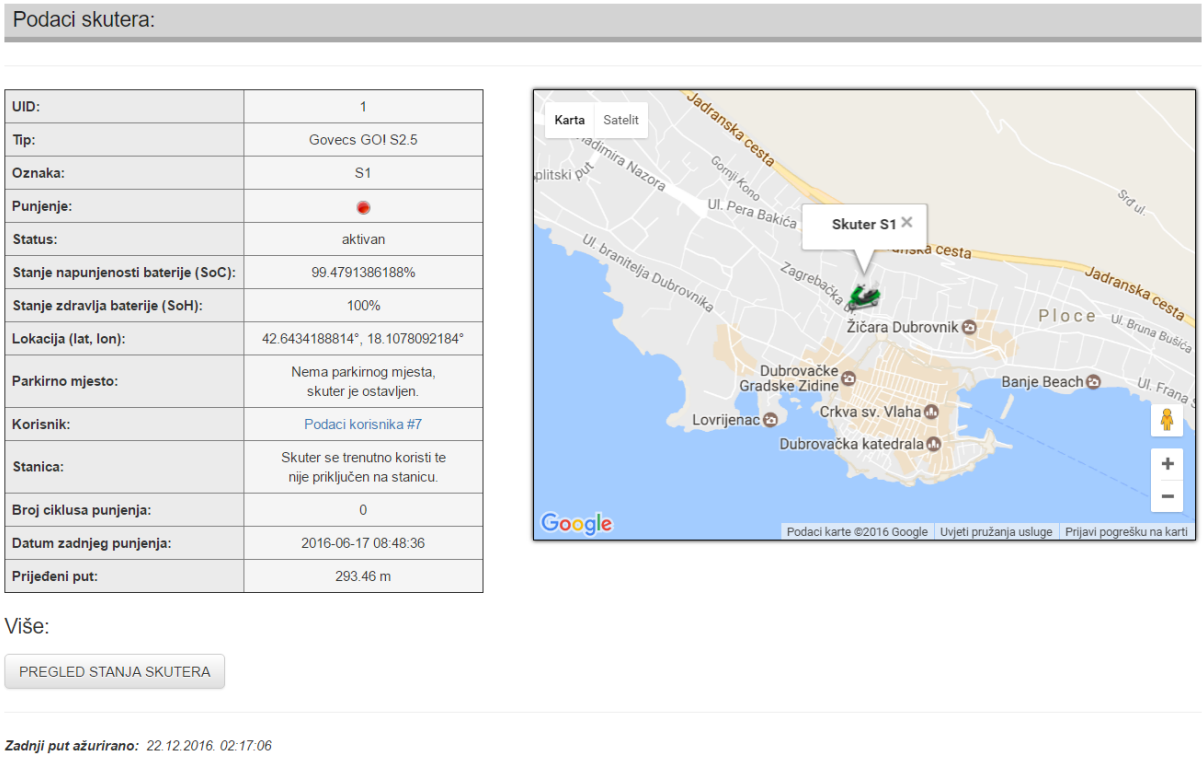


Slika 12: Prikaz pregleda e-mopeda aplikacije centra vođenja

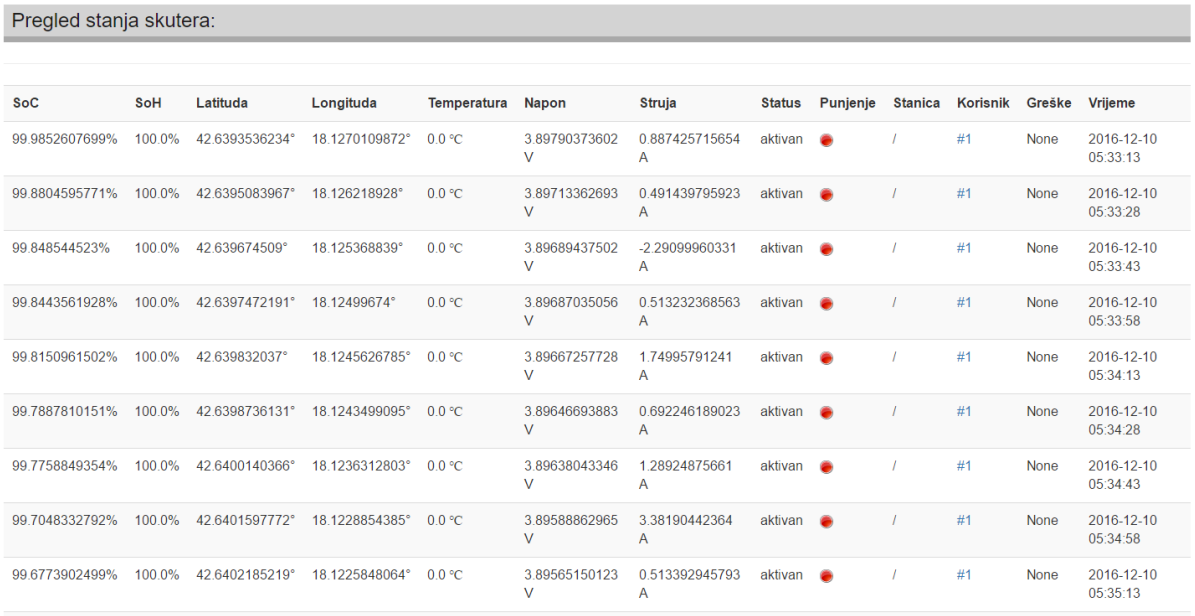
3.2.3. Detaljni prikaz stanja e-mopeda

Na ekranu detaljnog prikaza stanja određenog e-mopeda prikazani su svi njegovi podaci kao što je UID, tip, oznaka, status punjenja, status aktivnosti, stanje napunjenosti baterije (*SoC*), stanje zdravlja baterije (*SoH*), trenutne **GPS** koordinate, trenutno parkirno mjesto, trenutni korisnik ili stanica, ukupni prijeđeni put, broj ciklusa punjenja te datum zadnjeg punjenja koji se automatski osvježavaju tijekom vremena (Slika 13). Također je prikazana i *Google* mapa

koja sadrži prikaz e-mopeda na trenutnoj lokaciji i gumb za pregled svih njegovih stanja, odnosno podataka koji su periodički slani prema centru vođenja od početka njegove uporabe do sadašnjeg trenutka (Slika 14).



Slika 13: Prikaz detaljnog pregleda podataka učitanog e-mopeda aplikacije centra vođenja



SoC	SoH	Latituda	Longituda	Temperatura	Napon	Struja	Status	Punjenje	Stanica	Korisnik	Greške	Vrijeme
99.9852607699%	100.0%	42.6393536234°	18.1270109872°	0.0 °C	3.89790373602 V	0.887425715654 A	aktivan		/	#1	None	2016-12-10 05:33:13
99.8804595771%	100.0%	42.6395083967°	18.126218928°	0.0 °C	3.89713362693 V	0.491439795923 A	aktivan		/	#1	None	2016-12-10 05:33:28
99.848544523%	100.0%	42.639674509°	18.125368839°	0.0 °C	3.89689437502 V	-2.29099960331 A	aktivan		/	#1	None	2016-12-10 05:33:43
99.8443561928%	100.0%	42.6397472191°	18.12499674°	0.0 °C	3.89687035056 V	0.513232368563 A	aktivan		/	#1	None	2016-12-10 05:33:58
99.8150961502%	100.0%	42.639832037°	18.1245626785°	0.0 °C	3.89667257728 V	1.74995791241 A	aktivan		/	#1	None	2016-12-10 05:34:13
99.7887810151%	100.0%	42.6398736131°	18.1243499095°	0.0 °C	3.89646693883 V	0.692246189023 A	aktivan		/	#1	None	2016-12-10 05:34:28
99.7758849354%	100.0%	42.6400140366°	18.1236312803°	0.0 °C	3.89638043346 V	1.28924875661 A	aktivan		/	#1	None	2016-12-10 05:34:43
99.7048332792%	100.0%	42.6401597772°	18.1228854385°	0.0 °C	3.89588862965 V	3.38190442364 A	aktivan		/	#1	None	2016-12-10 05:34:58
99.6773902499%	100.0%	42.6402185219°	18.1225848064°	0.0 °C	3.89565150123 V	0.513392945793 A	aktivan		/	#1	None	2016-12-10 05:35:13

Slika 14: Prikaz pregleda stanja e-mopeda aplikacije centra vođenja (otvara se pritiskom na gumb PREGLED STANJA SKUTERA prikazanog na slici 13)

4. MODELIRANJE I SIMULIRANJE SUSTAVA DIJELJENJA ELEKTRIČNIH MOPEDA

U ovom poglavlju detaljno je opisan simulacijski sustav 24-satnog dijeljenja e-mopeda, a koji uključuje: model e-mopeda kao najbitnije značajke sustava čija se stanja moraju osvježavati konstantno tijekom simulacije, te stohastičke modele koji se koriste za generiranje vremena pristupa sustavu i određivanje sljedeće destinacije korisnika. Naposljetku je detaljno opisana tehnika programiranja simulacijskog algoritma cjelokupnog sustava.

4.1. Modeliranje električnog mopeda

4.1.1. Karakteristike odabranog e-mopeda

Pri odabiru prikladnog električnog mopeda nužan uvjet bio je da spada u B kategoriju vozila što znači da mu nazivna neto-snaga elektromotora i najveća konstrukcijska brzina ne smiju biti veći od 4 kW i 45 km/h. Kao najbolja alternativa pokazao se električni moped *Govecs GO! S2.5* zbog zadovoljavajućih performansi, relativno povoljne cijene, mogućnosti nadogradnje **GPS** lokatora, kamere i sučelja za registraciju na vozilu, te mogućnosti pristupa internim podacima i paljenja bez ključa preko **CAN**-a (*engl. Controller Area Network*)[5]. Tehničke specifikacije e-mopeda dane su u tablici 10.

Slika 15: E-moped Govecs GO! S2.5 [17]

Tablica 10: Tehnički podaci e-mopeda Govecs GO! S2.5 [17]

Maksimalna brzina	45 km/h
Domet	60-90 km
Kočnice	Prednje i stražnje hidrauličke disk kočnice
Gume	130/60-R13 sprijeda i straga
Prednja vilica	Hidraulična teleskopska vilica
Stražnji ovjes	Mono amortizer
Baterija	Litij
Napon	72 V
Težina baterije	30kg
Baterija (estimirani životni vijek)	50 000 km
Punjač	72 V punjač na vozilu, 110-240 V (50 / 60Hz)
Vrijeme punjenja	4-5 h, cca. 2 sata za punjenje do 80%
Motor	Motor bez četkica, PMAC s remenski prijenosom
Moment	54 Nm
Težina	cca. 120 kg (uklj. bateriju)
Međuosovinski razmak kotača	1 300 mm
Visina sjedala	790 mm
Prijenosni kapacitet	2 putnika / max. 180 kg
Garancija	24 mjeseca
Emisija	Nula
Cijena	37.596,00 kn (bez PDV-a)

4.1.2. Kvazistatički model e-mopeda

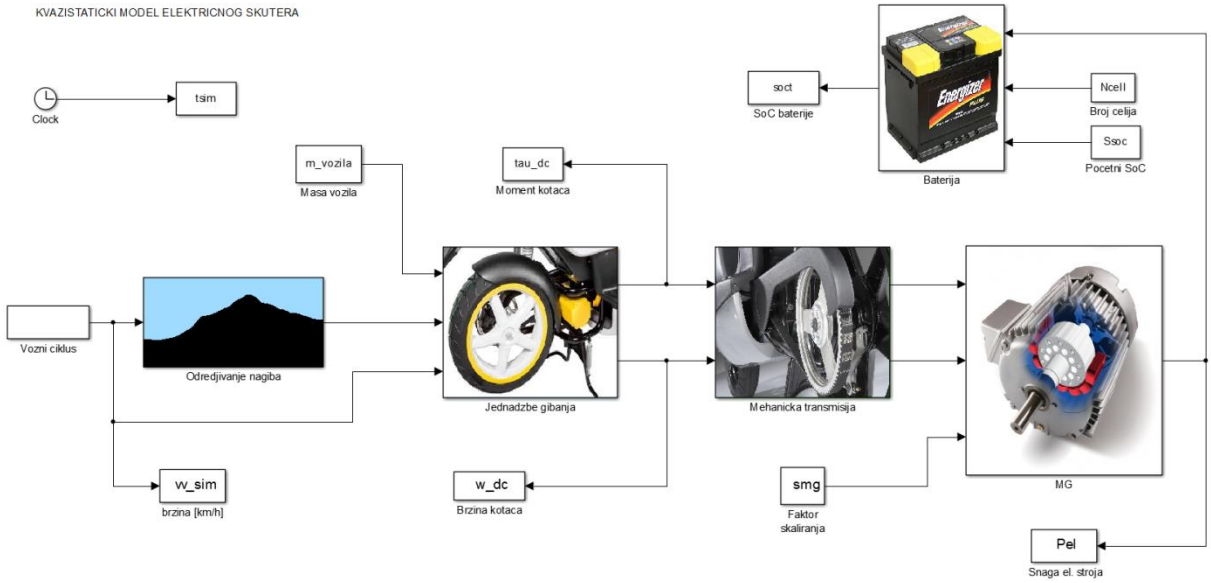
Kvazistatički (*backward*) model električnog mopeda izrađen je na temelju modificiranog modela konvencionalnog električnog autobusa. Model sadrži jedan ulaz, odnosno brzinu vozila (v_v) u diskretnom vremenskom koraku, dok je izlaz iz modela stanje napunjenosti baterije (*SoC*) koji predstavlja jedinu varijablu stanja u modelu. Moment na kotaču (τ_L) i kutna brzina kotača (ω_L) računaju se prema sljedećim jednadžbama (blok *"Jednadžbe gibanja"* - Slika 16, *Simulink* model bloka - Slika 17)[18]:

$$\tau_L = r\left[m_v\left(\frac{dv_v}{dt} + g(\sin\alpha + R_o\cos\alpha)\right) + 0.5\rho_{zraka}C_dA_fv_v^2\right] \quad \textbf{(1)}$$

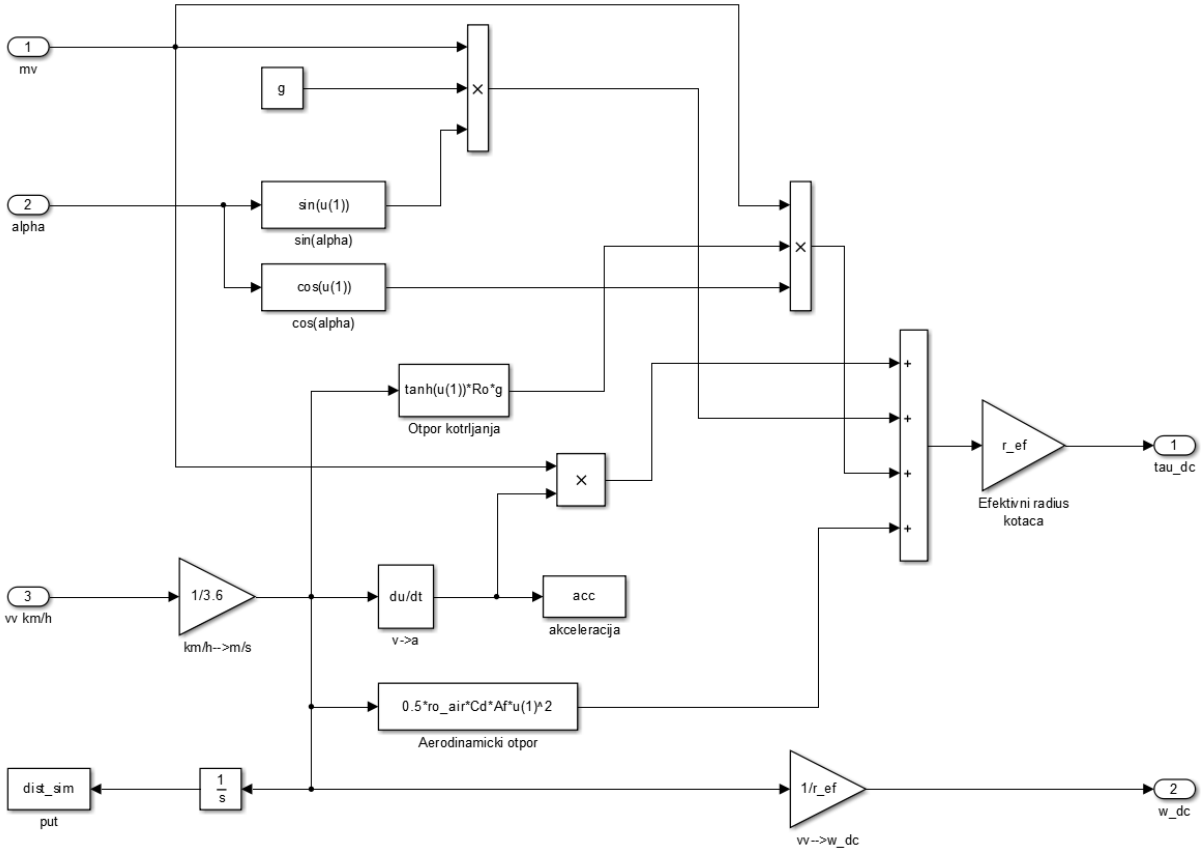
$$\omega_L = \frac{v_v}{r} \quad \textbf{(2)}$$

Jednadžbe (1) i (2) uključuju sljedeće poznate i procijenjene parametre: masa vozila (m_v), efektivni polumjer gume (r), aerodinamički koeficijent otpora zraka (Cd), koeficijent otpora kotrljanja guma (R_o), površina frontalnog poprečnog presjeka vozila (A_f) , gustoća zraka sa standardnom vrijednošću od 1,225 kg/m^3 (ρ_{zraka}) i gravitacijsko ubrzanje (g). Ukupna masa

vozila korištena u simulacijama izračunava se kao zbroj mase praznog mase vozila ($m_{v,\,praznog}$) i procijenjene prosječne mase putnika ($m_{v,\,putnika}$).



Slika 16: Model električnog mopeda u Simulinku



Slika 17: Implementacija jednadžba gibanja u Simulinku

Model električnog stroja definiran je mapom gubitaka snage ovisnom o trenutnim vrijednostima kutne brzine i okretnog momenta, te krivuljom maksimalnog momenta. Okretni

moment stroja (τ_{mg}) i kutna brzina (ω_{mg}) vezani su s brzinom vozila (v_v) i momentom tereta (τ_L) sljedećim kinematskim jednadžbama:

$$\tau_{mg} = \frac{\tau_L}{i_o \eta_t} \tag{3}$$

$$\omega_{mg} = i_o \omega_L = i_o \frac{v_v}{r} \tag{4}$$

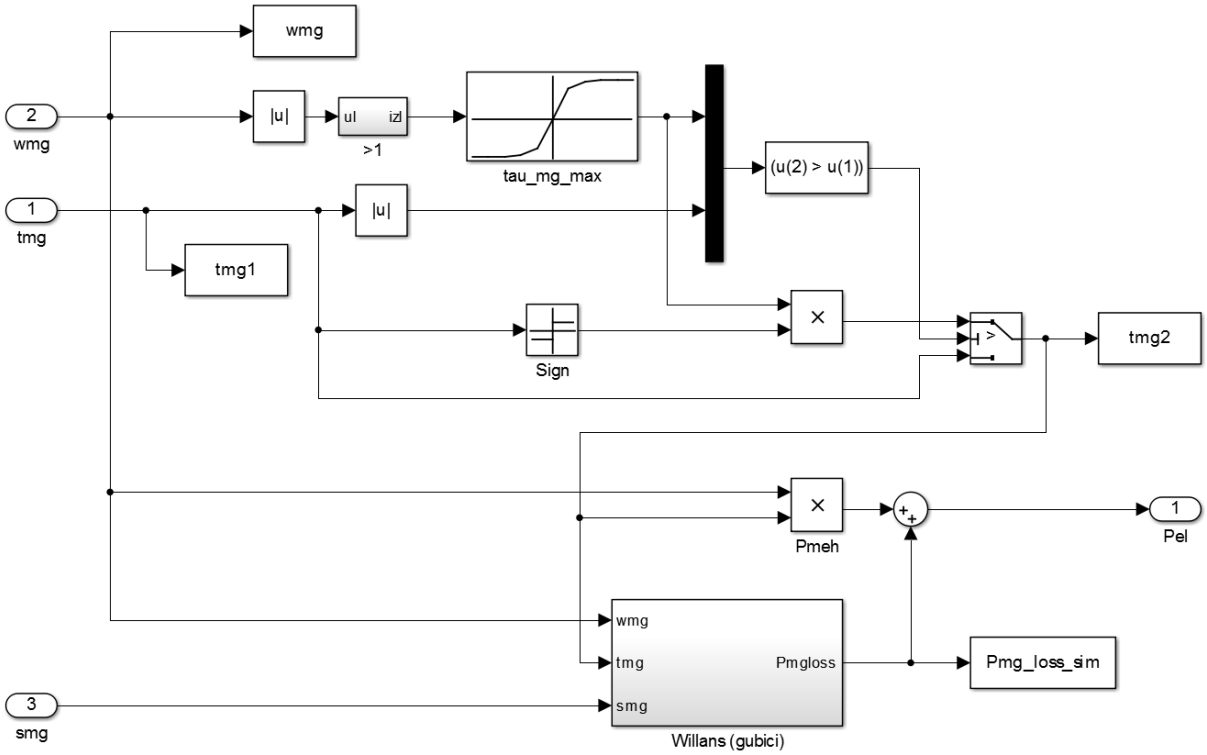
U jednadžbama (3) i (4) i_o označava prijenosni omjer remenice preko koje se prenosi snaga s kotača na električni stroj, a η_t stupanj iskoristivosti mehaničke transmisije. Ukupna snaga električnog stroja računa se kao zbroj mehaničke snage (P_{meh}) i gubitaka snage ($P_{mg,gub}$) prema sljedećim jednadžbama:

$$P_{meh} = \tau_{mg} \omega_{mg} \tag{5}$$

$$P_{mg,gub} = \begin{cases} \frac{1}{s_{mg}} c_1 \tau_{mg}^2 + c_2 \tau_{mg} + c_3 s_{mg} \text{ za } P_{mg,gub} > P_{mg,gub\_min} \\ P_{mg,gub\_min} \leq P_{mg,gub\_min} \end{cases} \tag{6}$$

$$P_{el} = P_{meh} + P_{mg,gub} \tag{7}$$

gdje s_{mg} predstavlja faktor skaliranja elektromotora, c_1, c_2, c_3 varijabilne koeficijente aproksimacije, P_{meh} mehaničku snagu električnog stroja, $P_{mg,gub}$ gubitke snage električnog stroja, P_{mg,gub_min} minimalne gubitke snage električnog stroja, a P_{el} ukupnu snagu električnog stroja (vidi detalje u sljedećem odjeljku).

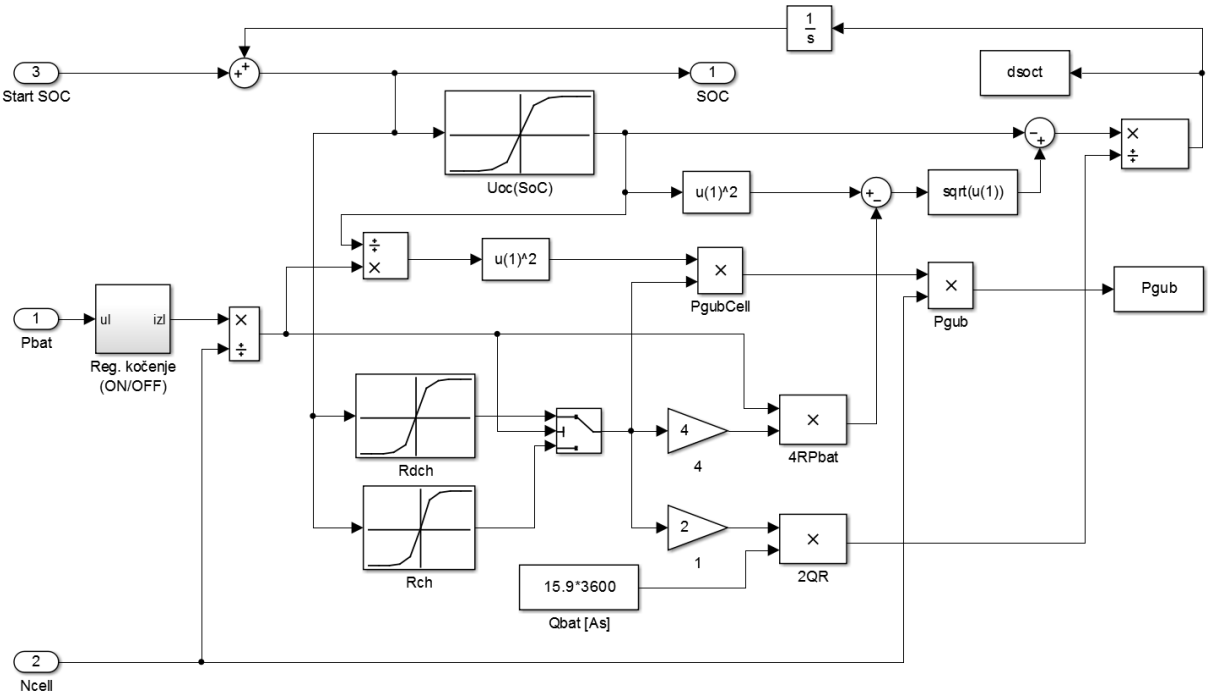


Slika 18: Implementacija modela električnog stroja u Simulinku

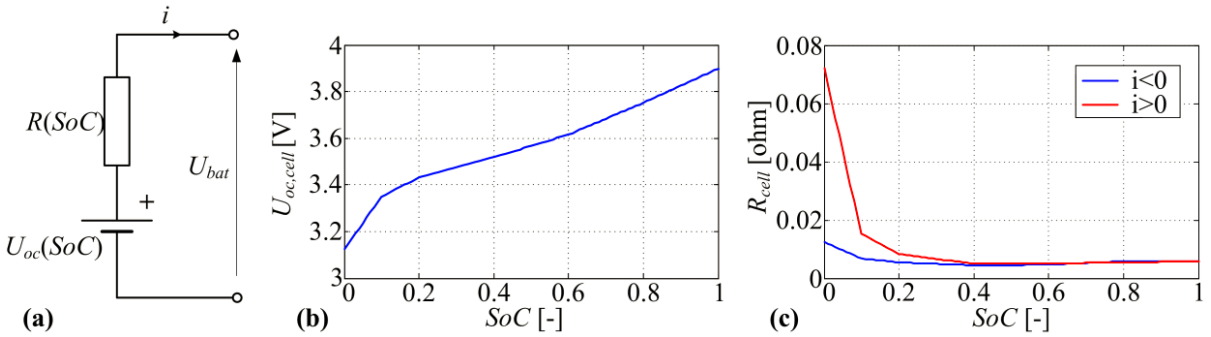
Baterija je modelirana na bazi nadomjesnog električnog kruga (Slika 20a) zbog svoje relativne jednostavnosti naspram drugih kompleksnijih modela, te predstavlja osnovu za izvođenje sljedeće jednadžbe stanja baterije:

$$\dot{SoC}(t) = \frac{\sqrt{U_{OC}^2\big(SoC(t)\big) - 4R(SoC(t),i)P_{bat}(t)} - U_{OC}\big(SoC(t)\big)}{2Q_{\max}R(SoC(t),i)} \qquad \textbf{(8)}$$

gdje $Q_{\max}$ predstavlja ukupni kapacitet baterije, a SoC ($0 \leq SoC \leq 1$) varijablu stanja modela. Ovisnost napona otvorenog kruga baterije U_{oc} o SoC-u za jednu Li-Ion baterijsku ćeliju prikazana je na slici 20b. Ovisnost o SoC-u također vrijedi i za unutarnji otpor jedne ćelije baterije R (Slika 20c). Varijabla P_{bat} označava snagu baterije, a po iznosu je jednaka snazi električnog stroja P_{el} (7).



Slika 19: Implementacija modela baterije u Simulinku



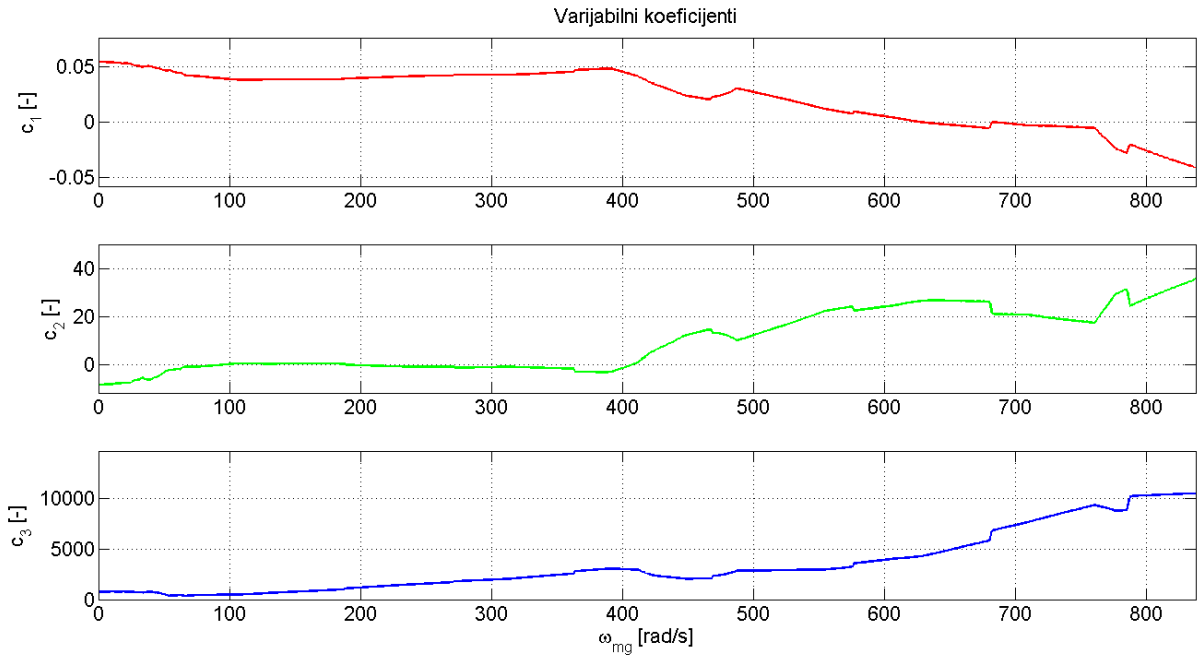
Slika 20: Nadomjesni električni krug baterije (a), napon otvorenog kruga jedne ćelije (b), unutrašnji otpor jedne ćelije u ovisnosti o SoC-u (c) [18]

4.1.3. Modeliranje gubitaka električnog stroja korištenjem Willans metode aproksimacije

Skaliranje modela elektromotora električnog autobusa s ciljem prilagodbe njegovih karakteristika ograničenjima e-mopeda ostvaruje se korištenjem skalabilnih karakteristika koje trebaju što točnije predvidjeti efikasnost elektromotora s obzirom na dimenzijske parametre s_{mg} i s_{fc}. Dakle, mapa gubitaka elektromotora prethodno korištena u modelu električnog autobusa prilagođena je elektromotoru električnog mopeda.

Jedna od mogućnosti bila bi interpolacija vrijednosti pojedinog parametra kroz veliki skup snimljenih podataka za različito dimenzionirane elektromotore [19]. Međutim, takav pristup uglavnom nije moguć s obzirom na ograničen pristup mjernim podacima za različite tehnologije i dimenzije strojeva pa se stoga uglavnom koristi Willans-ovo pravilo aproksimacije koje je temeljeno na opisu karakteristike gubitaka u stroju [20][21].

Također je, uz Willans-ovu aproksimaciju, moguće gubitke snage elektromotora $P_{mg,gub}$ dovoljno dobro opisati polinomom drugog stupnja [22] uključujući dimenzijski parametar s_{mg} na način kako prikazuje jednadžba (6), gdje su c_1, c_2 i c_3 koeficijenti ovisni o brzini vrtnje elektromotora (ω_{mg}). Navedene koeficijente moguće je odrediti za svaku točku osi brzine vrtnje iz originalne (nazivne) karakteristike gubitaka koristeći metodu najmanjih kvadrata. Tako dobivene vrijednosti prikazuje Slika 21.



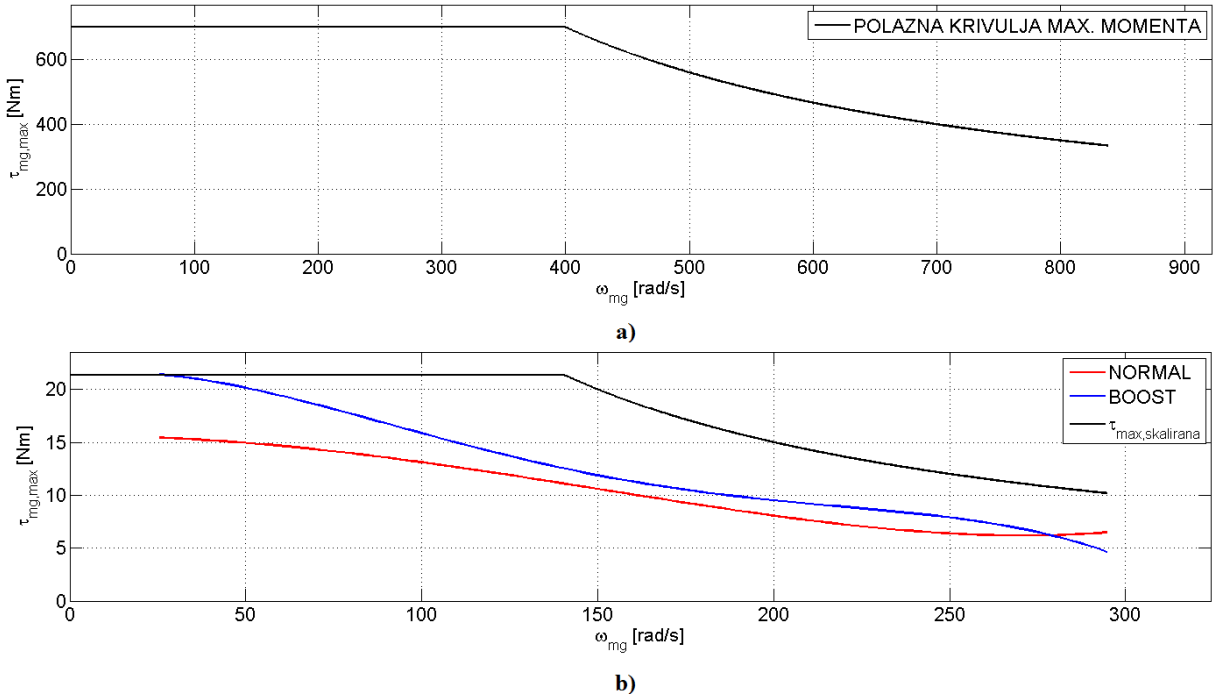
Slika 21: Koeficijenti aproksimacije polinomom drugog stupnja

Vrijednosti parametara s_{fc} i s_{mg} za skalirani model elektromotora iznose redom 0,3516 i 0,0305, a dobiveni su dijeljenjem maksimalne kutne brzine i momenta e-mopeda s

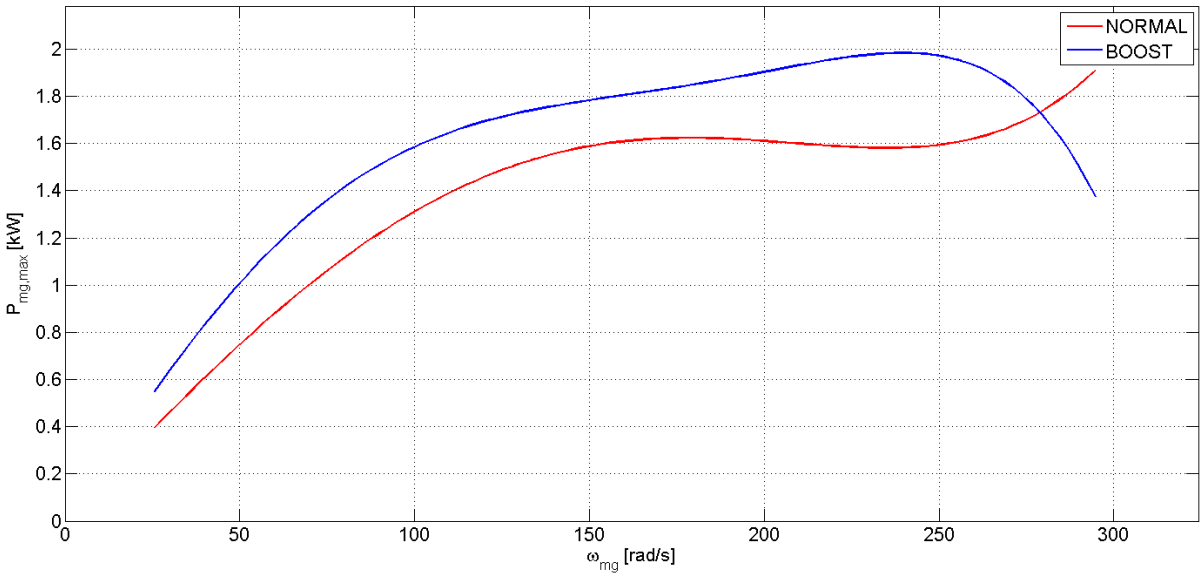
maksimalnom kutnom brzinom i momentom električnog autobusa. Za krivulju maksimalnog momenta pretpostavljena je linearna ovisnost o dimenzijskom parametru s_{mg} kako slijedi:

$$\tau_{mg,max}(\omega_{mg}) = s_{mg} \cdot \tau_{mg0,max}(\omega_{mg}) \quad \textbf{(9)}$$

Prilikom izračuna gubitaka snage elektrčinog stroja također je uzeta u obzir njena minimalna vrijednost (P_{mg,gub_min}) čiji je iznos jednak 5 % maksimalne snage e-mopeda. Krivulje maksimalnih momenata u ovisnosti o kutnoj brzini kotača prije i nakon skaliranja elektromotora, zajedno s krivuljama maksimalnih momenta za NORMAL i BOOST način rada e-mopeda koje su dobivene eksperimentalnim putem, prikazane su na slici 22, dok su krivulje maksimalnih snaga za oba režima rada prikazane na slici 23.



Slika 22: Polazna krivulja maksimalnog momenta (a), skalirana krivulja maksimalnog momenta, zajedno s krivuljama maksimalnih momenata na kotaču za NORMAL i BOOST režim rada e-mopeda (b)



Slika 23: Krivulje maksimalnih snaga e-mopeda za NORMAL i BOOST režim rada

4.2. Stohastičko modeliranje i generiranje prijava korisnika na stanicu

Stohastički pristupi korisnika sustavu modelirani su kontinuiranim funkcijama definiranim na vremenskom intervalu od jednog dana, a koje predstavljaju očekivani broj pristupa korisnika u svakom vremenskom trenutku tokom dana. Pritom su pristupi korisnika sustavu posebno modelirani za stanice u centru, te posebno za stanice na periferiji. Ukoliko stanica pripada periferiji, očekivani broj korisnika biti će najveći tijekom jutarnjih sati, a nešto manji u večernjim satima, dok za stanice koje pripadaju centru vrijedi obratno (vidi sliku Slika 24).

Funkcije su definirane kombiniranjem triju funkcija gustoće slučajne varijable normalne razdiobe s različitim parametrima (μ_i, σ_i) od kojih je svaka definirana kako slijedi:

$$N_i(x \mid, \mu_i, \sigma_i) = \frac{1}{\sqrt{2\sigma_i^2 \pi}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}} \qquad i = 1, 2, \ldots, 6 \tag{10}$$

gdje u ovom slučaju x predstavlja trenutno vrijeme, μ matematičko očekivanje (odnosno vremensku vrijednost centra krivulje), a σ standardno odstupanje (ovim parametrom je određen oblik, odnosno širina razdiobe). Na ovaj način se modeliranje očekivanog broja pristupa sustavu svodi na odabir nekoliko lako podesivih parametara. Vrijednosti odabranih parametara svake od šest krivulja normalnih razdioba dane su u tablici 11.

Tablica 11: Vrijednosti parametara funkcija gustoće slučajene varijable normalnih razdioba i očekivanog broja korisnika

	μ [h]	σ [h]	Očekivani broj korisnika
N_1	8	2	2.0
N_2	14	4	1.0

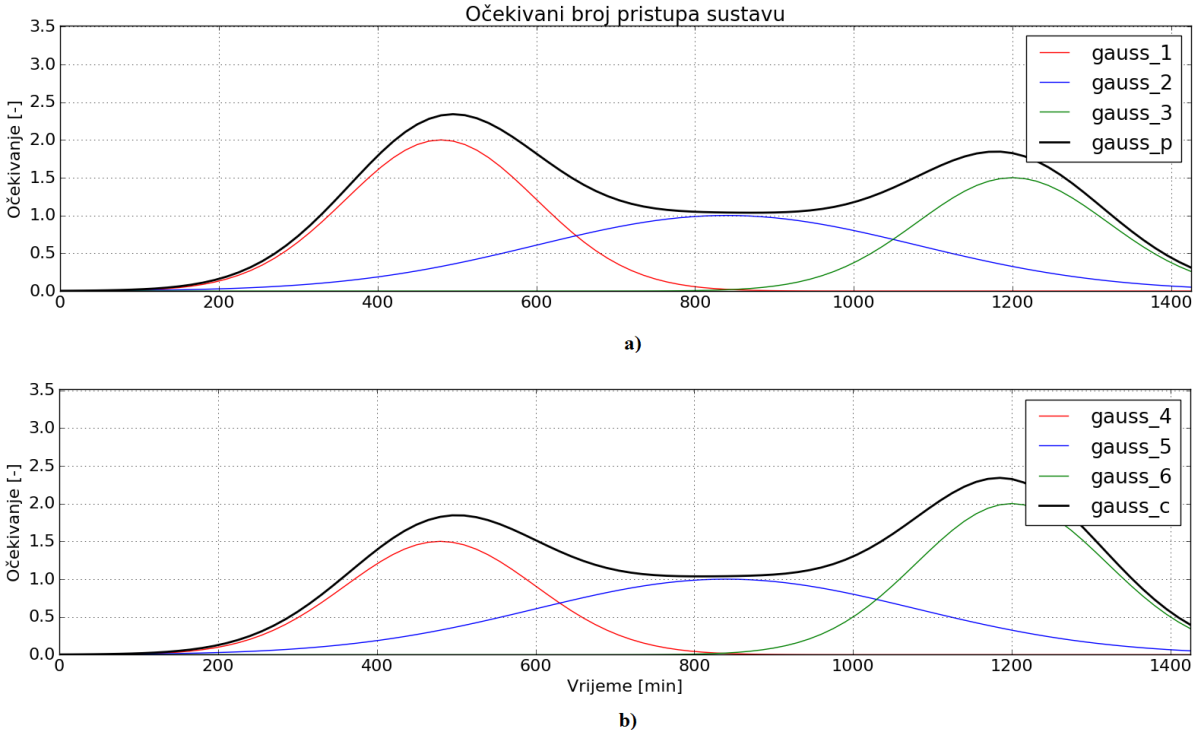
N_3	20	2	1.5
N_4	8	2	1.5
N_5	14	4	1.0
N_6	20	2	2.0

Ukupne vrijednosti funkcija *gauss_c* i *gauss_p* jednake su sumama triju Gaussovih krivulja pomnoženima s faktorima skaliranja koji se temelje na vršnoj vrijednosti očekivanih brojeva korisnika za pripadni vremenski interval:

$$f_p(x) = \sum_{i=1}^{n} \theta_i N_i(x \mid, \mu_i, \sigma_i)\,; \qquad n = 3 \tag{11}$$

$$f_c(x) = \sum_{i=n+1}^{m} \theta_i N_i(x \mid, \mu_i, \sigma_i)\,; \qquad m = 6 \tag{12}$$

$$\theta_i = \frac{očekivani\ broj\ korisnika}{\max(f_i(x \mid, \mu_i, \sigma_i))}; \qquad i = 1, 2, \ldots, 6 \tag{13}$$



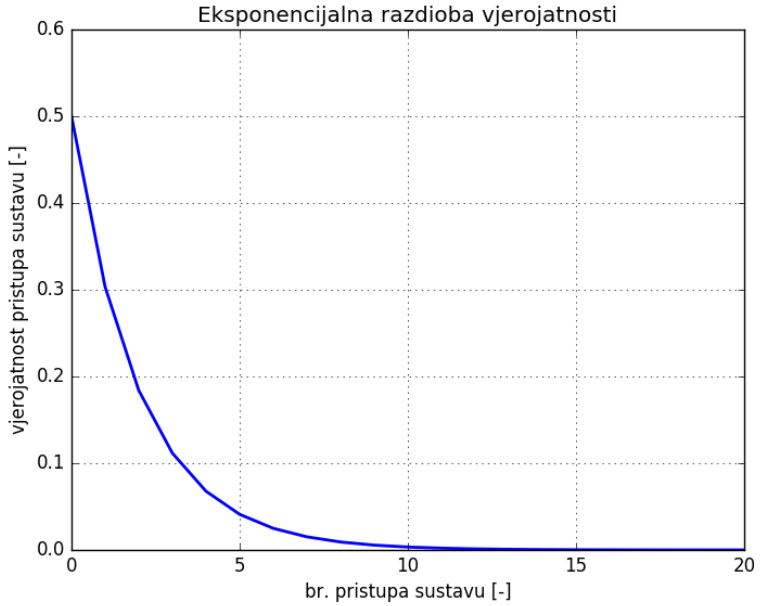
Slika 24: Očekivani brojevi pristupa sustavu za stanicu na periferiji (a), stanicu u centru (b)

Potom se za svaki vremenski interval od 15 minuta unutar jednog dana, vjerojatnost pristupa određenog broja korisnika modelira eksponencijalnom razdiobom kako slijedi:

$$E(x) = \lambda e^{-\lambda x}; \qquad \lambda = \frac{1}{OBK} \tag{14}$$

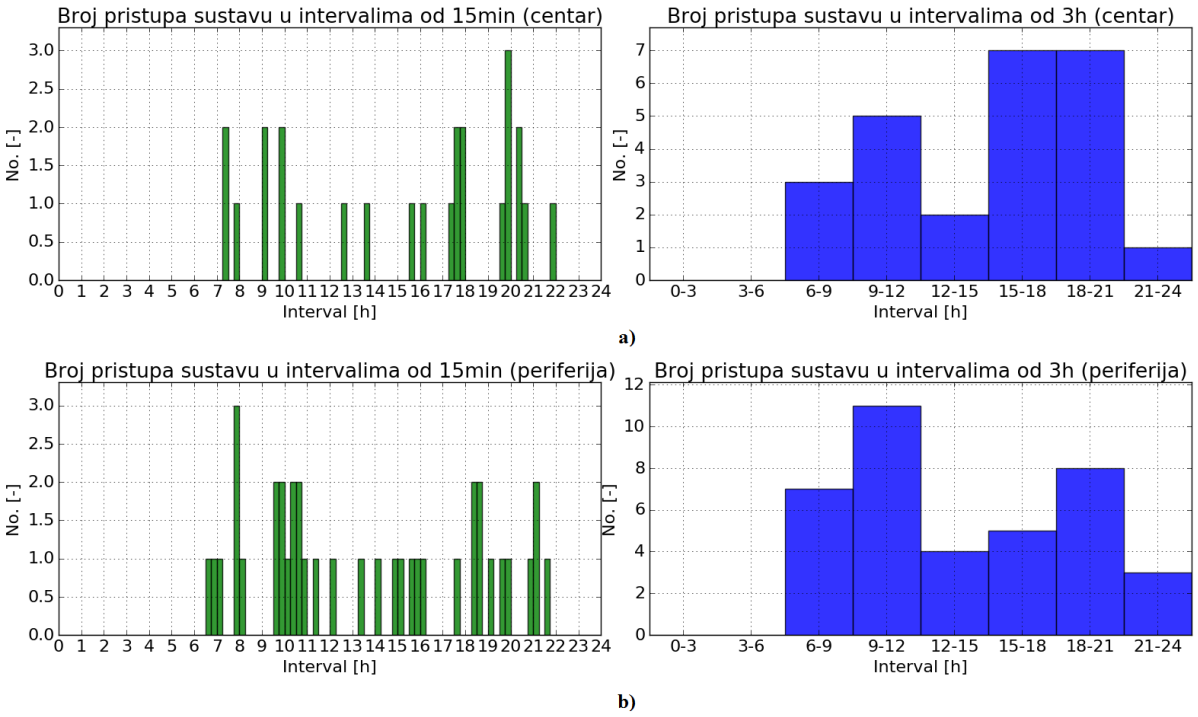
gdje *x* predstavlja broj korisnika, *E* vjerojatnost pristupa sustavu, a *OBK* očekivani broj korisnika u pripadnom vremenskom intervalu koji je definiran funkcijom *gauss_c* ili *gauss_p*,

ovisno za koju stanicu se određuje broj pristupa (u centru ili na periferiji; vidi jednadžbe (11) i (12) te sliku Slika 24). Primjer razdiobe vjerojatnosti pristupa sustavu za *OBK*=2 prikazan je na slici 25.



Slika 25: Vjerojatnosti pristupa sustavu temeljeni na eksponencijalnoj razdiobi

Zatim su uz pomoć generatora slučajnog broja čija je vrijednost u intervalu [0, 1] za svaku stanicu generirani pristupi sustavu kroz vremenski interval od 24 sata (Slika 26).



Slika 26: Primjer generiranih pristupa sustavu za stanicu koja se nalazi u centru (a) i na periferiji (b)

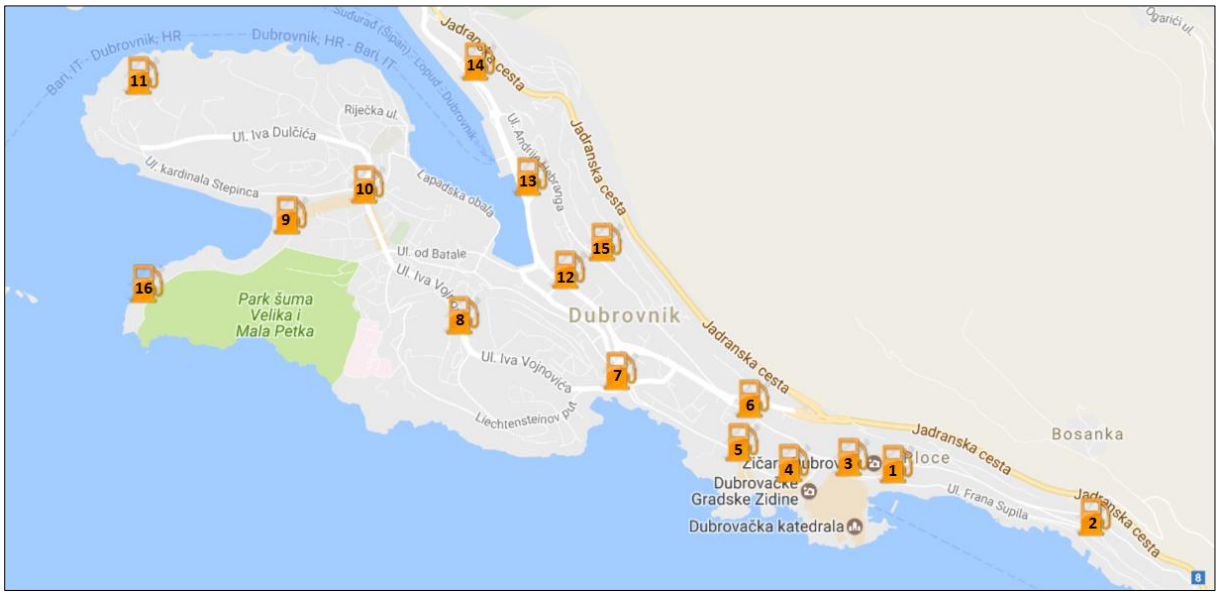
Kao što je vidljivo, za stanicu u centru broj pristupa korisnika sustavu u jutarnjim satima manji je nego u kasnim podnevnim satima, dok za stanicu koja se nalazi u periferiji vrijedi obratno, odnosno generirane razdiobe pristupa sustavu vjerno reprezentiraju zadani model (Slika 24).

4.3. Stohastičko modeliranje i generiranje odabira destinacije

Kako bi sustav odredio sljedeću destinaciju korisnika, najprije treba znati na kojoj stanici je došlo do preuzimanja e-mopeda, pripada li ta stanica centru ili periferiji, te na kojoj lokaciji (geografskoj širini i dužini) se ta stanica nalazi. U ovom radu, lokacije stanica predložene su od strane grada Dubrovnika, te je svakoj pridodano svojstvo pripadnosti centru ili periferiji, te određeni težinski koeficijent (proizvoljno odabran). Sva svojstva stanica, zajedno s njihovim koordinatama dana su u tablici 12, dok su njihove pozicije na karti Dubrovnika prikazane na slici 27.

Tablica 12: Svojstva i lokacije svih stanica sustava

UID	Lokacija	Geografska širina/dužina		Težinski koeficijent	Pripadnost centru
1	Ploče	42.642032	18.113304	1.00	Ne
2	Petra Krešimira IV	42.639346	18.127050	0.90	Ne
3	Buža Tenis	42.642384	18.110222	1.10	Ne
4	Pile	42.642080	18.106057	1.15	Ne
5	Gradac kod IUC-a	42.643158	18.102617	1.10	Ne
6	Zagrebačka Ulica	42.645358	18.103362	1.50	Da
7	Put Iva Vojnovića/Bana Jelačića	42.646777	18.094120	1.25	Da
8	Put Iva Vojnovića	42.649618	18.083247	2.00	Da
9	Uvala Lapad/Lapadski Dvori	42.654742	18.071281	1.50	Da
10	Kralja Tomislava/Pošta Lapad	42.656296	18.076660	1.80	Da
11	Babin Kuk/Argosy	42.661833	18.061005	1.45	Ne
12	Vukovarska	42.651938	18.090553	1.85	Da
13	Gruž/Obala I. Pavla II	42.656683	18.087927	1.30	Ne
14	Gruž / Kantafig	42.662543	18.084308	1.30	Ne
15	Gruž/A. Hebranga/S. Cvijića	42.653341	18.093079	1.75	Da
16	Hotel Dubrovnik Palace	42.651235	18.061331	1.45	Ne



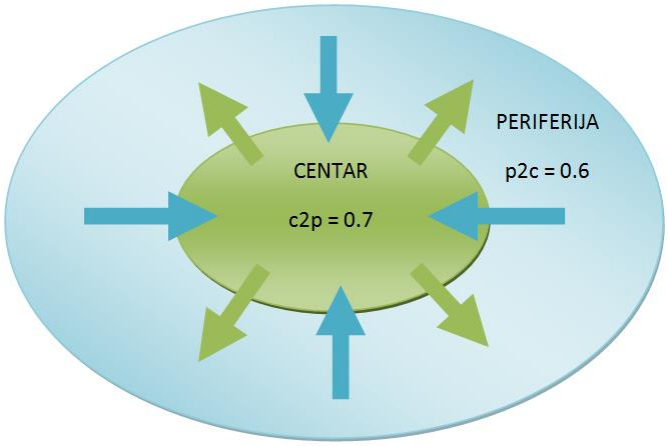
Slika 27: Pozicije stanica na karti Dubrovnika

Jednom kada sustav registrira pristup sustavu najprije se određuje lokacija pripadne stanice i njena pripadnost. Ukoliko stanica pripada centru uz pomoć generatora slučajnih brojeva koji generira slučajan broj iz intervala [0, 1], određuje se da li korisnik ostaje u centru, ili putuje prema periferiji. Pritom, ako korisnik ostaje u centru, kao potencijalne destinacije izdvajaju se samo one stanice koje pripadaju centru (osim trenutne), odnosno u slučaju ako korisnik napušta centar izdvajaju se samo stanice periferije. Destinacija se na sličan način odabire i u slučaju da se trenutna stanica (polazišna točka) nalazi na periferiji.

Dakle, postoje dva koeficijenta prijelaza (Slika 28) čije su vrijednosti u ovom radu odabrane proizvoljno zbog nepostojanja statističkih podataka iz kojih bi se njihova vrijednost mogla izračunati, a to su:

- **c2p** (*engl. center to periphery*) - vjerojatnost prijelaza iz centra u periferiju
- **p2c** (*engl. periphery to center*) - vjerojatnost prijelaza iz periferije u centar

Nakon izdvajanja mogućih destinacija izračunava se zračna udaljenost od trenutne stanice do svake izdvojene primjenom određenih metoda koje se temelje na projekciji koordinata danih u obliku geografske širine i dužine u Kartezijev koordinatni sustav. Dakle, sve formule koje će biti navedene vrijede za izračune koji se temelje na sfernom obliku Zemlje (ignorirajući elipsoidne efekte), što daje dovoljno točne rezultate u većini slučajeva. U stvari, Zemlja je vrlo blago elipsoidna pa korištenje sfernog modela daje greške obično do 0,3 % [23].



Slika 28: Koeficijenti prijelaza iz centra u periferiju i obratno

Pri računanju udaljenosti između dvije točke, u sljedećim formulama korišten je efektivni polumjer zemlje (R) čija približna vrijednost iznosi 6378,137 km. Također, φ označava geografsku širinu (*engl. latitude*), a λ geografsku dužinu (*engl. longitude*).

I. Haversin metoda [23]:

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot sin^2\left(\frac{\Delta\lambda}{2}\right) \quad \textbf{(15)}$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \quad \textbf{(16)}$$

$$d = R \cdot c \quad \textbf{(17)}$$

gdje c predstavlja kutnu udaljenost u radijanima, d udaljenost između dvije točke, dok je a kvadrat pola dužine luka (*engl. chord*) između zadanih točaka. U ovom radu, za izračun udaljenosti između dviju stanica, korištena je ova metoda zbog najveće preciznosti pri malim udaljenostima.

II. **SLC** (***S**pherical **L**aw of **C**osines*) metoda [23]:

$$d = \text{acos}(\sin\varphi_1 \cdot \sin\varphi_2 + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \cos\Delta\lambda) \cdot \text{R} \quad \textbf{(18)}$$

III. **EA** (***E**quirectangular **A**pproximation*) metoda [23]:

$$x = \Delta\lambda \cdot \cos\left(\frac{\varphi_1 + \varphi_2}{2}\right) = \Delta\lambda \cdot \cos\varphi_m \quad \textbf{(19)}$$

$$y = \Delta\varphi \quad \textbf{(20)}$$

$$d = R \cdot \sqrt{\text{x}^2 + \text{y}^2} \quad \textbf{(21)}$$

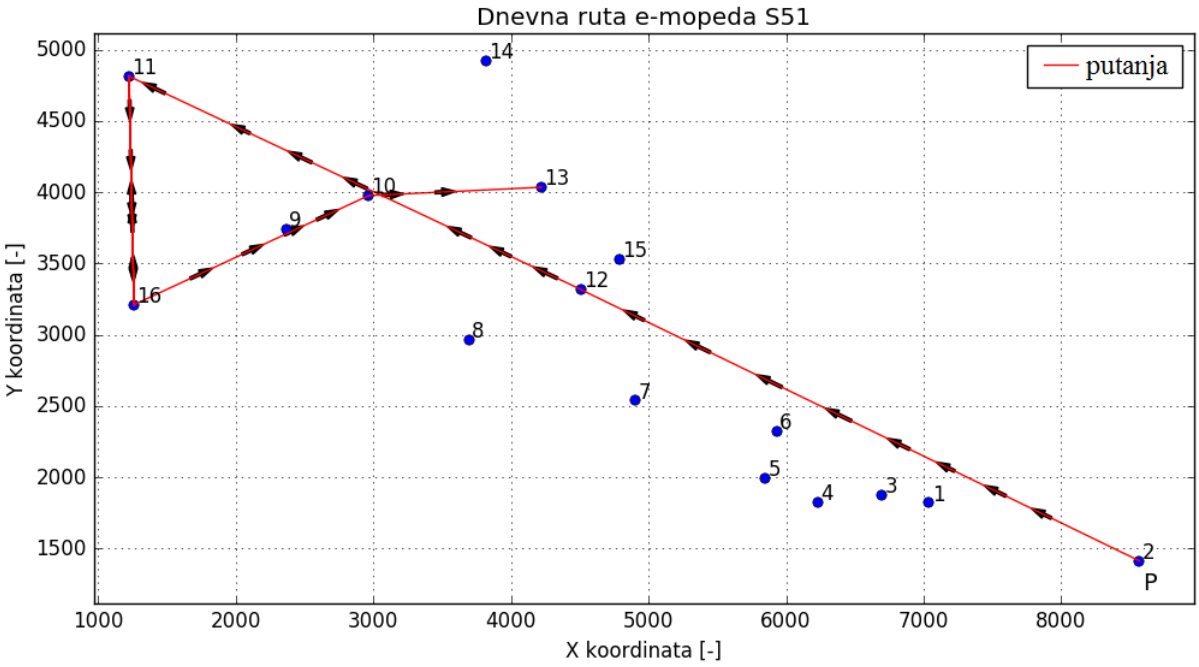
Jednom kada su izračunate udaljenosti između trenutne stanice i svih mogućih destinacija, za svaku od stanica izračunava se vrijednost funkcije težine (f_w) kako slijedi:

$$f_{w,i}(x) = w_i \cdot \lambda e^{-\lambda x}; \qquad \lambda \in R \tag{22}$$

gdje x označava udaljenost između stanica, w težinski koeficijent stanice, a λ neki proizvoljno odabrani realni broj (u ovom slučaju λ=1). Pritom je odabrana funkcija eksponencijalnog tipa zbog toga što njena vrijednost znatno opada s porastom udaljenosti, pa će stoga najbliže stanice imati najveću vjerojatnost da će biti odabrane kao sljedeća destinacija. Vjerojatnost odabira svake destinacije dobiva se normiranjem težinskih funkcija (22) kako slijedi:

$$p_i = \frac{f_{w,i}}{\sum_{i=1}^{n} f_{w,i}} \tag{23}$$

gdje n označava broj stanica. Na kraju se uz pomoć generatora slučajnog broja čija se vrijednost nalazi u intervalu [0, 1] određuje sljedeća destinacija korisnika. Primjer generirane dnevne rute e-mopeda, gdje plave točke označavaju pozicije stanica, a P polaznu stanicu, prikazuje Slika 29.



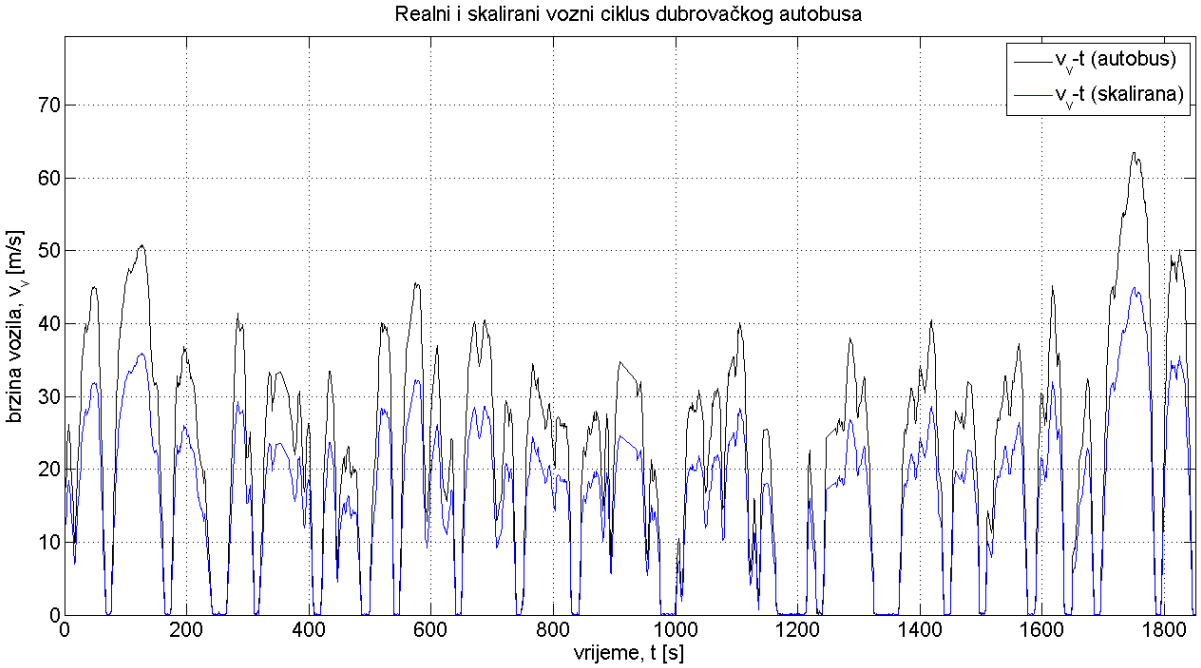
Slika 29: Primjer generirane dnevne rute korisnika

4.4. Obrada snimljenih voznih ciklusa

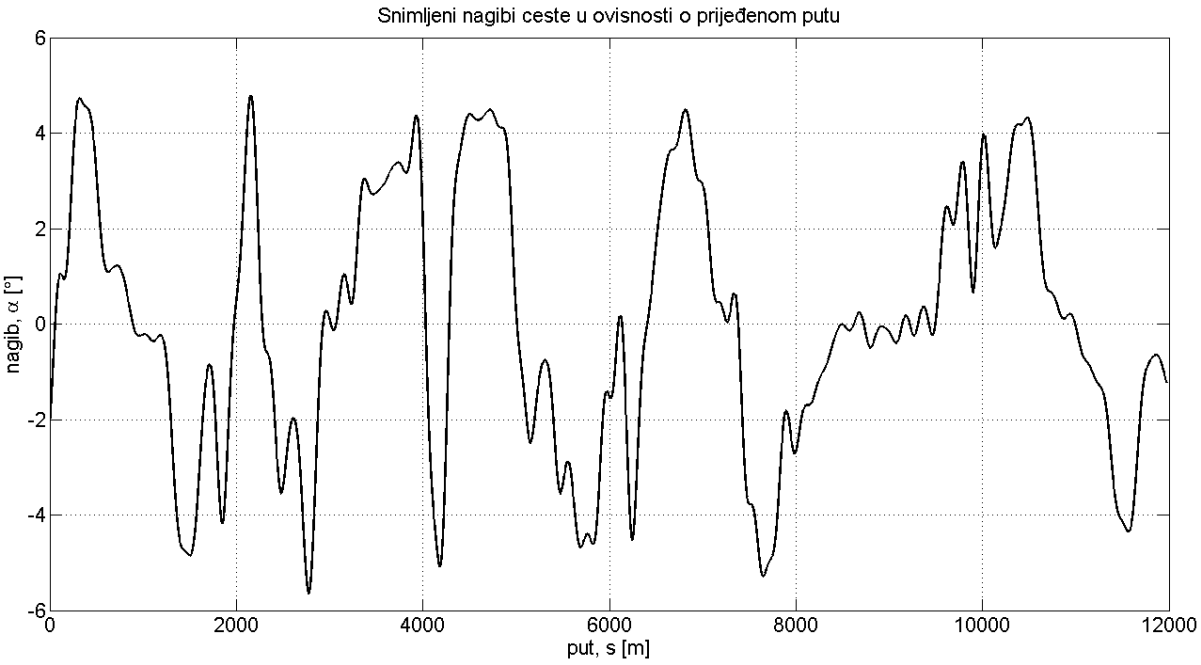
Zbog nedostatka snimljenih podataka voznih ciklusa električnih mopeda u ovom radu su korišteni korigirani vozni ciklusi snimljeni na jednoj ruti dubrovačkih autobusa s periodom uzorkovanja od jedne sekunde. Snimljeni podaci sadrže ovisnosti brzine o proteklom vremenu, te ovisnosti nagiba ceste o prijeđenom putu. Pošto neke od snimljenih brzina autobusa znatno odstupaju od maksimalne brzine e-mopeda koja iznosi 45 km/h najprije je bilo potrebno skalirati vozne cikluse tako da ograničenje brzine bude zadovoljeno.

$$v_{v,sk} = v_v \cdot \frac{v_{\max,mopeda}}{\max(v_v)} \quad \textbf{(24)}$$

Jedan skalirani vozni ciklus dubrovačkog autobusa prikazuje Slika 30, dok ovisnost nagiba ceste o prijeđenom putu prikazuje Slika 31.



Slika 30: Prikaz realnog i skaliranog voznog ciklusa dubrovačkog autobusa



Slika 31: Prikaz snimljenih nagiba u ovisnosti o prijeđenom putu

Nakon skaliranja voznih ciklusa po brzini bilo je potrebno izvršiti ograničenje momenta na kotaču vozila primjenom krivulje ovisnosti maksimalnog momenta u odnosu na brzinu vozila

koja je dobivena eksperimentalnim putem. Detaljan postupak ograničavanja momenta opisan je u odjeljku 4.4.1.

4.4.1. Korekcija snimljenih voznih ciklusa ograničavanjem izlaznog momenta

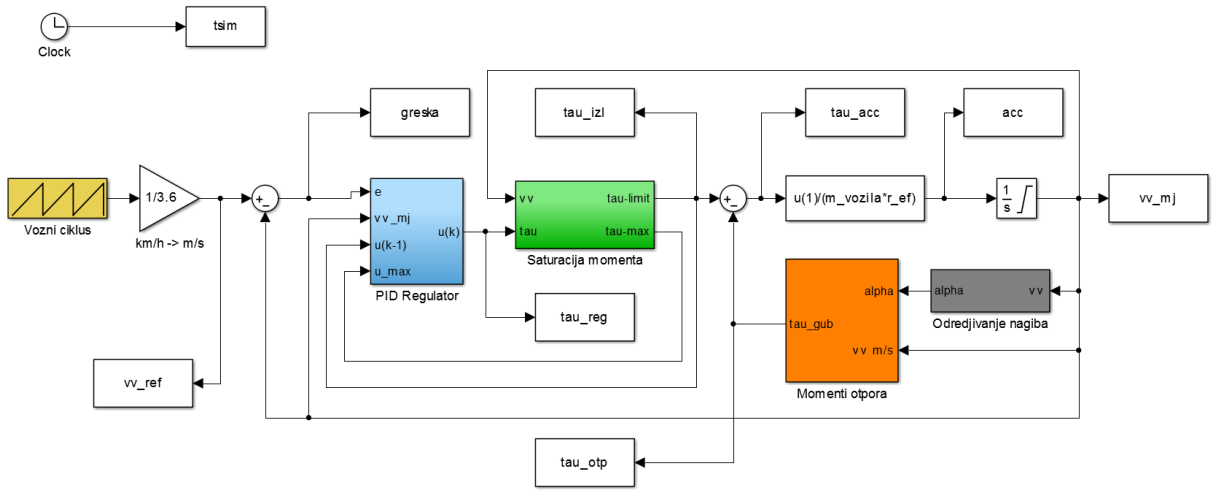
PID regulator skaliranih voznih ciklusa dubrovačkih autobusa modeliran je na temelju jednadžbe (1) tako da upravljački signal (izlaz regulatora) predstavlja željeni moment na kotaču vozila (τ_L) od kojega se oduzimaju gubici momenta (τ_{gub}), odnosno moment aerodinamičkog otpora zraka (τ_{aero}), moment otpora trenja kotrljanja (τ_{kot}) i moment uslijed kosine (τ_{alpha}), čime se dobiva moment ubrzanja vozila (τ_a) iz kojega se nakon integriranja po vremenu dobiva brzina vozila (v_v).

$$\tau_L = \tau_a + \tau_{gub} = \tau_a + \tau_{alpha} + \tau_{kot} + \tau_{aero} \quad \textbf{(25)}$$

$$\tau_a = \tau_L - \tau_{gub} = m_v r_{ef} \frac{dv_v}{dt} \quad \textbf{(26)}$$

$$v_v = \int \frac{1}{m_v r_{ef}} \tau_a \, dt \quad \textbf{(27)}$$

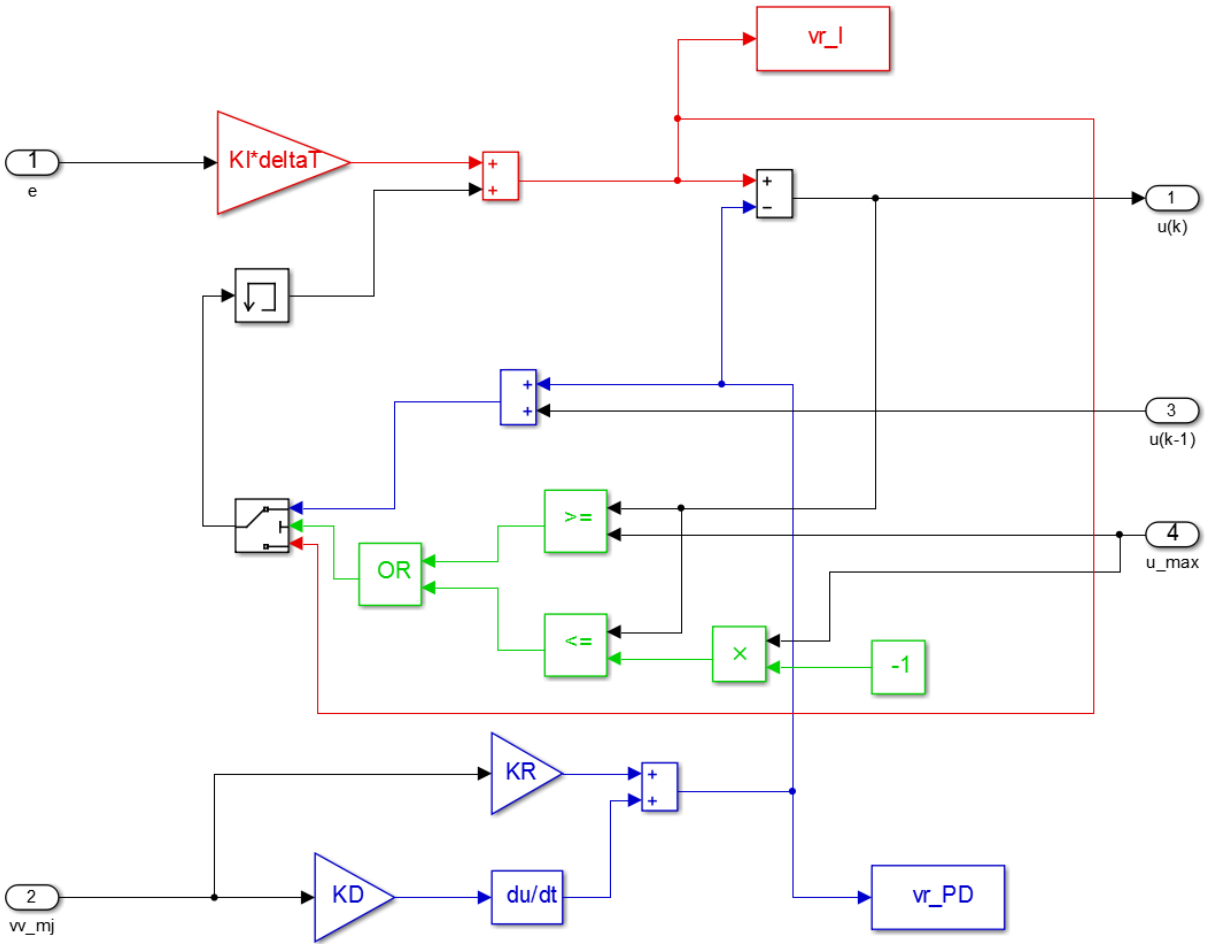
Ulazi u regulator su redom (Slika 32): regulacijska pogreška (e) koja je jednaka razlici između referentne i mjerene brzine vozila, mjerena brzina ($v_{v,mj}$), zadnja vrijednost izlaznog momenta ($u(t\text{-}T)$), te vrijednost maksimalnog momenta za trenutnu brzinu ($u_{\max}$).



Slika 32: Implementacija Simulink modela za korekciju snimljenih voznih ciklusa ograničavanjem izlaznog momenta

Zbog integriranja regulacijske pogreške e, stanje integratora može postići vrlo velike iznose pa stoga regulator uz limitiranje izlaza uključuje i *"reset anti-windup"* intervenciju (Slika 33): ukoliko je razlika proporcionalno-derivirajućeg (PD) i integrirajućeg (I) djelovanja izvan izlaznog raspona regulatora, stanje integratora se resetira na vrijednost koja odgovara sumi pripadajućeg limita i PD djelovanja [24].

$$u(t) = u_{limit} = u_I(t) - u_{PD}(t) \quad \textbf{(28)}$$



Slika 33: Implementacija PID regulatora s "reset anti-windup" intervencijom u Simulinku

4.4.2. Optimiranje parametara PID regulatora primjenom Nelder-Mead algoritma

Nelder-Mead metodom, odnosno metodom fleksibilnog poliedra traži se minimum funkcije cilja *F(e(k), u(k))* koja je definirana kako slijedi [24]:

$$F\big(e(t),u(t)\big) = \frac{1}{M+1}\sum_{k=0}^{M} f_w(k)\big[e(k)^2 + \rho K_p^2\big(u(k) - u_0(k)\big)\big] \quad \textbf{(29)}$$

$$e(k) = v_{v,ref} - v_{v,mj} \quad \textbf{(30)}$$

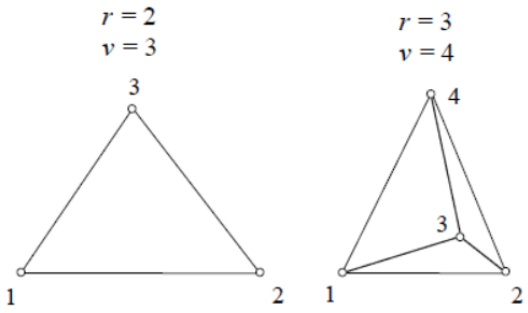
$$u_0(k) = \frac{v_{v,ref}}{K_p} \quad \textbf{(31)}$$

$$f_w(k) = S(k - W) \quad \textbf{(32)}$$

gdje je M broj uzoraka uzoraka unutar razmatranog vremenskog intervala, f_w vremenski prozor unutar kojeg se obavlja optimiranje, odnosno vremenski pomaknuta *step* funkcija, *e*

regulacijska pogreška, ρ težinski faktor, u upravljački signal (izlaz regulatora), K_p pojačanje procesa, a u_0 stacionarna vrijednost upravljačkog signala.

Metoda fleksibilnog poliedra (MFP) je numerička metoda programiranja (optimiranja), koja se temelji na metodi traženja pomoću fleksibilnog poliedra, koji mijenja svoj položaj i oblik tijekom postupka iteriranja. MFP je ustvari modifikacija *simplex* metode, koja je razvijena za probleme bez ograničenja. *Simplex* je geometrijska figura s $v = r + 1$ vrhova, formirana u r-dimenzionalnom prostoru, gdje r predstavlja broj stupnjeva slobode problema. Za slučaj problema s tri stupnja slobode, poliedar ima 4 vrha – tetraedar, a za probleme s 2 stupnja slobode trokut s 3 vrha. Za slučaj problema s jednim stupnjem slobode, također se koristi trokut s 3 vrha [25].



Slika 34: Regularni simplex s 2 i 3 stupnja slobode [25]

Od početnog istostraničnog *simplex*-a, ova metoda u daljnjim koracima mijenja njegove dimenzije tako da se vrijednosti ciljne funkcije u njegovim vrhovima približavaju minimumu ciljne funkcije. Proces se prekida kada poliedar postane dovoljno malen ili kada vrijednosti u vrhovima postanu izjednačene. Algoritam metode sastoji se u računanju vrijednosti ciljne funkcije u svakom vrhu, te se izdvoje one s najboljom (najmanjom) i najlošijom (najvećom) vrijednosti ciljne funkcije, zatim se računa centar poliedra bez vrha u kojem je najlošija vrijednost ciljne funkcije, te srednje rastojanje od vrhova poliedra do centra, da bi se nakon toga potražio novi vrh u okolini koja ima bolju vrijednost ciljne funkcije. To se ostvaruje kombinacijom četiri moguće operacije istraživanja okoline [25]:

1. **REFLEKSIJOM** najlošijeg vrha kroz centar, prema formuli za k-ti korak:

$$\boldsymbol{x}_{refleksije}^{(k)} = x_C^{(k)} + \alpha\left(\boldsymbol{x}_C^{(k)} - \boldsymbol{x}_{najlosiji}^{(k)}\right) \qquad \textbf{(33)}$$

   gdje je koeficijent refleksije $\alpha \geq 0$, uz preporučenu vrijednost $\alpha = 1$.

2. **EKSPANZIJOM**, odnosno dodatnim udaljavanjem novog vrha od najlošijeg vrha:

$$\boldsymbol{x}_{ekspanzije}^{(k)} = \boldsymbol{x}_C^{(k)} + \gamma\left(\boldsymbol{x}_{refleksije}^{(k)} - \boldsymbol{x}_C^{(k)}\right), \qquad \gamma \geq 0 \qquad \textbf{(34)}$$

gdje γ predstavlja koeficijent ekspanzije, a preporučena vrijednost je $\gamma = 2$. Po završetku ekspanzije uspoređuju se vrijednosti u reflektiranom i ekspandiranom vrhu, te se kao novi vrh odabire onaj u kojemu ciljna funkcija poprima bolju vrijednost.

3. **KONTRAKCIJOM** ako je funkcija cilja u reflektiranoj točki lošija, nego u bilo kojem vrhu poliedra:

$$\boldsymbol{x}_{kontrakcije}^{(k)} = \boldsymbol{x}_{C}^{(k)} + \beta\left(\boldsymbol{x}_{najlosiji}^{(k)} - \boldsymbol{x}_{C}^{(k)}\right), \qquad 0 \leq \beta \leq 1 \tag{35}$$

gdje β predstavlja koeficijent kontrakcije, a preporučuje se vrijednost $\beta = 0.5$.

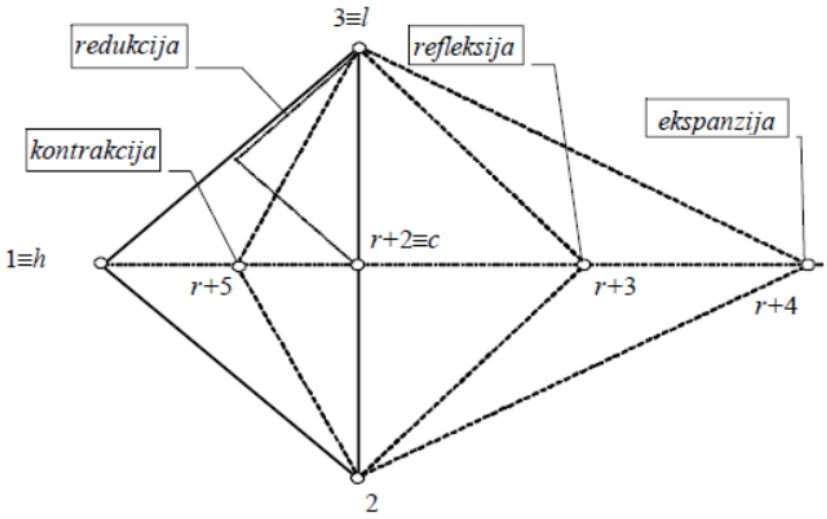
4. **REDUKCIJOM** dimenzija poliedra nakon kontrakcije, u slučaju da je:

$$F\left(\boldsymbol{x}_{kontrakcije}^{(k)}\right) > F\left(\boldsymbol{x}_{najlosiji}^{(k)}\right) \tag{36}$$

Kod primjene na probleme bez ograničenja ($r = n$), traženje se prekida u koraku u kojem je ostvaren izraz:

$$Tol(k) = \sqrt{\frac{1}{r+1}\sum_{i=1}^{r+1}\left(F_i^{(k)} - F_{r+2}^{(k)}\right)^2} \leq \varepsilon \tag{37}$$

gdje je ε neki unaprijed određen mali broj.



Slika 35: Prikaz operacija nad poliedrom s 3 vrha [25]

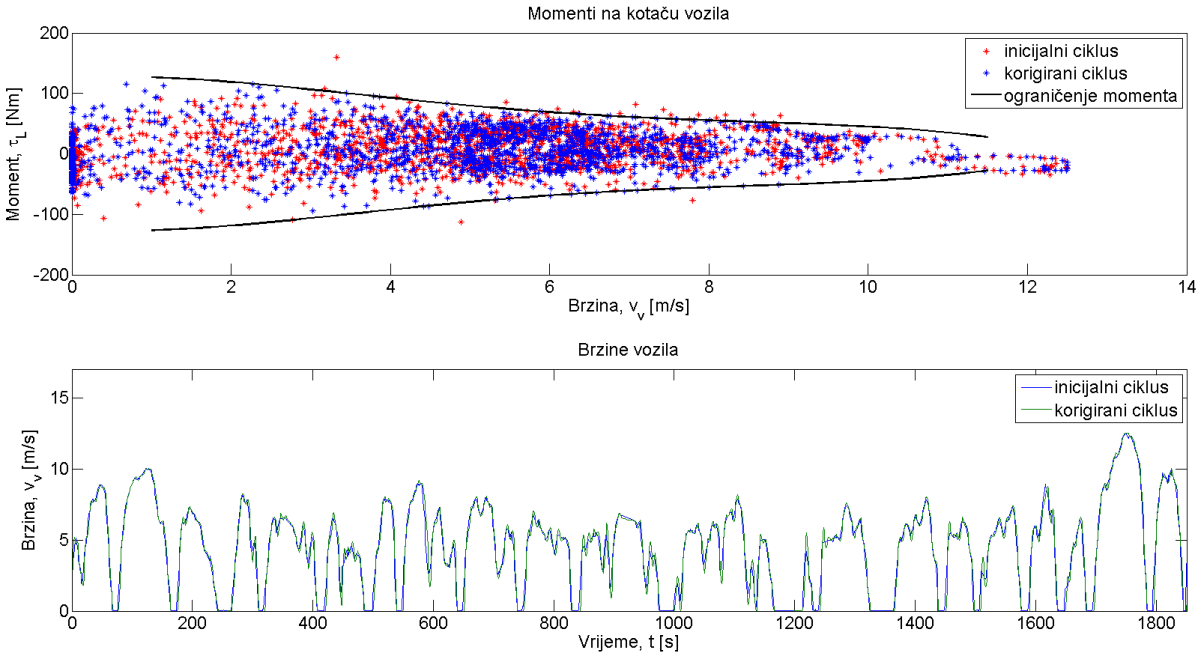
U ovom radu za potrebe optimiranja parametara PID regulatora, odnosno pronalaska minimuma funkcije cilja (29), korištena je *Matlab*-ova funkcija *fminsearch*. Inicijalne i dobivene optimalne vrijednosti pojačanja PID regulatora prikazuje Tablica 13.

Tablica 13: Inicijalne i optimalne vrijednosti pojačanja PID regulatora

	Br. voznog ciklusa	K_R	K_I	K_D
Inicijalne vrijednosti	-	10.0	10.0	10.0
Optimalne vrijednosti	1	14.9012	23.0316	0.0
	2	16.8803	15.7506	0.0
	3	16.4548	16.9435	0.0
	4	16.3189	17.2968	0.0
	5	39.8961	39.2083	0.0
	6	15.9784	18.2682	0.0
	7	16.8697	15.7933	0.0

4.4.3. Verifikacija korigiranih voznih ciklusa

Primjenom PID regulatora voznih ciklusa dobiveni su novi vozni ciklusi koji zadovoljavaju ograničenje maksimalnog momenta na kotaču e-mopeda (Slika 36). Kod nekih od dobivenih voznih ciklusa postoje sitna odstupanja od referentnih brzina upravo zbog toga što u određenim točkama nije bilo dovoljno "zaliha" momenta nužnog za prijelaz iz stanja trenutne u referentnu brzinu i akceleraciju (Slika 30). Ovako dobiveni korigirani vozni ciklusi dalje su korišteni kao ulazi u algoritam generiranja sintetičkih voznih ciklusa (vidi potpoglavlje 4.5).

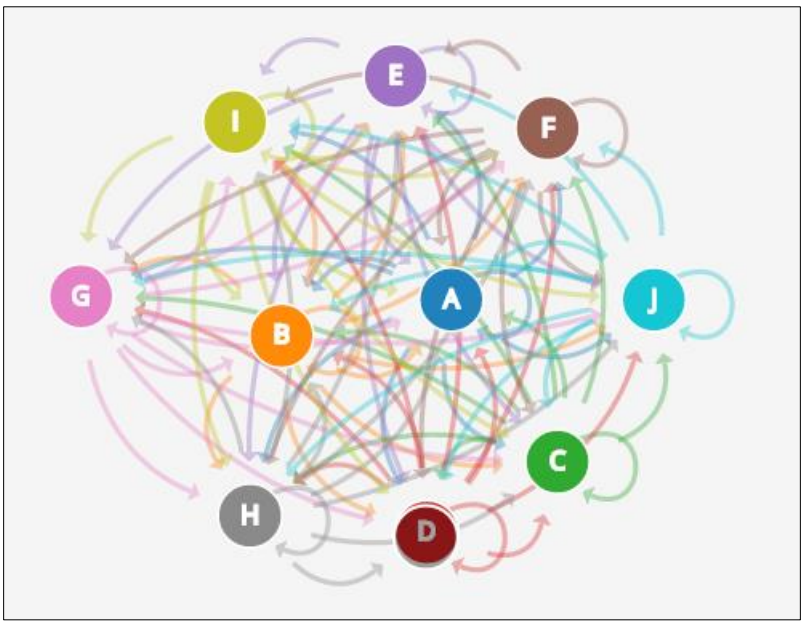


Slika 36: Momenti na kotaču vozila pri određenim brzinama (a), te referentni i korigirani vozni ciklus (b)

4.5. Stohastičko modeliranje i generiranje sintetičkih voznih ciklusa primjenom metode Markovljevih lanaca

4.5.1. Općenito o metodi Markovljevih lanaca

U matematici Markovljevi lanci, nazvani po Andreyu Markovu, predstavljaju niz stanja sustava. U svakome svakome diskretnom vremenskom trenutku (koraku) sustav može prijeći u neko novo stanje ili može ostati u istome stanju. Ako slijed stanja ima Markovljevo svojstvo to znači da je svako buduće stanje vremenski neovisno o svakome prijašnjem stanju. Drugim riječima, u teoriji vjerojatnosti stohastički proces ima Markovljevo svojstvo ako buduća raspodjela vjerojatnosti procesa, za dano trenutno stanje i sva prošla stanja, ovisi samo o trenutnome stanju i niti o jednom drugome prethodnom [26].



Slika 37: Primjer prikaza povezanosti stanja (A-F) Markovljevih lanaca [27]

Markovljev lanac je slijed slučajnih varijabla X_1, X_2, X_3, ..., X_n s Markovljevim svojstvo, a njegova matematička formulacija glasi:

$$\boldsymbol{P}(X_{n+1} = x \mid X_n = x_n, \dots, X_1 = x_1) = \boldsymbol{P}(X_{n+1} = x \mid X_n = x_n) \qquad \textbf{(38)}$$

Ako Markovljev lanac ima *k* mogućih stanja koje označavamo 1, 2, 3,...*k* onda se vjerojatnost da je sustav u stanju *j* u trenutku *t*+1 nakon što je bio u stanju *i* u trenutku *t* označava p_{ij} i zove se vjerojatnost prijelaza iz stanja *i* u stanje *j*. Matrica $\boldsymbol{P}$ = [*pij*] zove se matrica prijelaznih vjerojatnosti. Nužan uvjet je da zbroj elemenata u svakom retku matrice prijelaznih vrijednosti mora biti jednak 1:

$$\sum_{j=1}^{n} p_{ij} = 1 \quad \textbf{(39)}$$

Za svako stanje $j \in \{1, 2, ..., n\}$ prijelazna vjerojatnost p_{ij} predstavlja uvjetnu vjerojatnost da će se sustav naći u *j*-tom stanju ako se prethodno nalazio u *i*-tom stanju.

4.5.2. Proračun matrice prijelaznih vjerojatnosti

Na temelju korigiranih voznih ciklusa dobivenih u potpoglavlju 4.4 dobivena je matrica prijelaznih vjerojatnosti (***P***) prema sljedećem algoritmu:

- Kao stanja Markovljevog lanca odabrane su diskretne vrijednosti brzine i akceleracije. Kako je na raspolaganju samo brzina vozila (Slika 36b), ubrzanje vozila rekonstruirano je uz pomoć derivacije brzina kako slijedi:

  $$a(k) = \frac{v(k+1) - v(k)}{T} \quad \textbf{(40)}$$

  gdje je T = 1 s vrijeme diskretizacije.

- Generirano je polje stanja brzina v_{arr} s rasponom od 0-45 km/h uz rezoluciju od 0.1 km/h, gdje je maksimalna vrijednost raspona određena maksimalnom brzinom vozila.

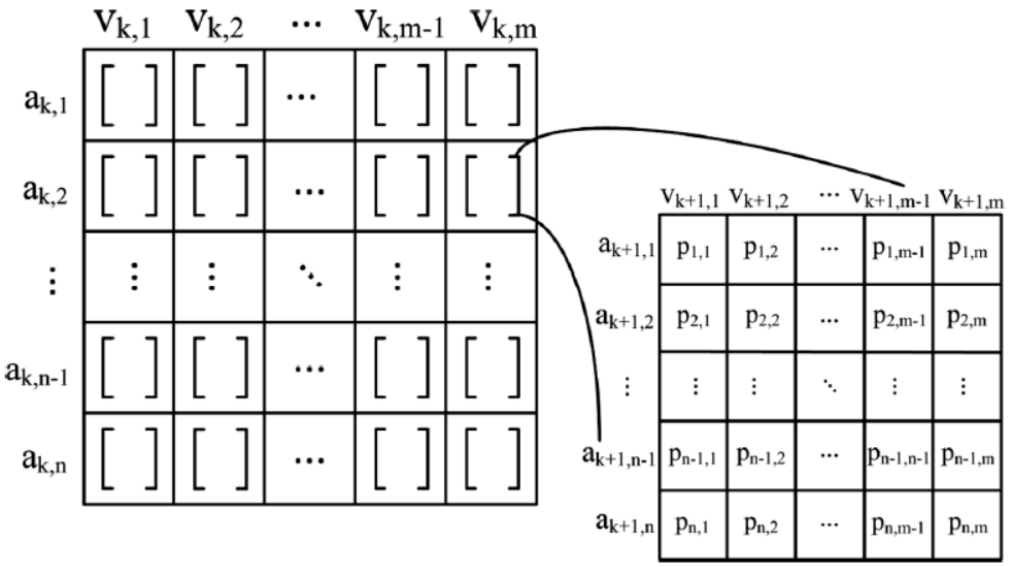
- Generirano je polje stanja akceleracija a_{arr} s rasponom od -2.0 - 2.0 m/s^2 uz rezoluciju od 0.4 m/s^2 zato što se većina snimljenih ubrzanja voznih ciklusa nalazi unutar ovog raspona akceleracija.

- Inicijalizirana je matrica prijelaznih vjerojatnosti $\boldsymbol{P} \in \boldsymbol{R}^4$ dimenzija $m\ x\ n\ x\ m\ x\ n$, gdje m = 450 predstavlja dimenziju polja v_{arr}, a n = 11 dimenziju polja a_{arr}. Vjerojatnosti prijelaza između diskretnih skupova stanja brzina i ubrzanja izračunati su kao [28]:

  $$p_{qr,ij} := \boldsymbol{P}\left(X_{k+1} = a_i, v_j \middle| X_k = a_q, v_r\right) \quad \textbf{(41)}$$

  a njihova distribucija je organizirana u formi matrice s pod-matricama, kao što prikazuje Slika 38. Indeksi redaka i stupaca matrice označavaju diskretne vrijednosti brzine i ubrzanja u trenutnom diskretnom vremenskom koraku označenom s *k*. Svaki element matrice sadrži pod-matricu koja sadržava vjerojatnosti prijelaza iz trenutnog stanja (a_q, v_r) u bilo koje druge stanje (a_i, v_j) u sljedećem (k + 1) diskretnom vremenskom koraku. Vrijednosti brzina i akceleracija u koraku *k* zaokružene su na njima najbliže vrijednosti unutar polja v_{arr} i a_{arr}. Budući da zbroj vjerojatnosti prijelaza iz trenutnog stanja u bilo koje drugo stanje treba biti jednako 1 $\left(\sum_i \sum_j p_{qr,ij} = 1\right)$,

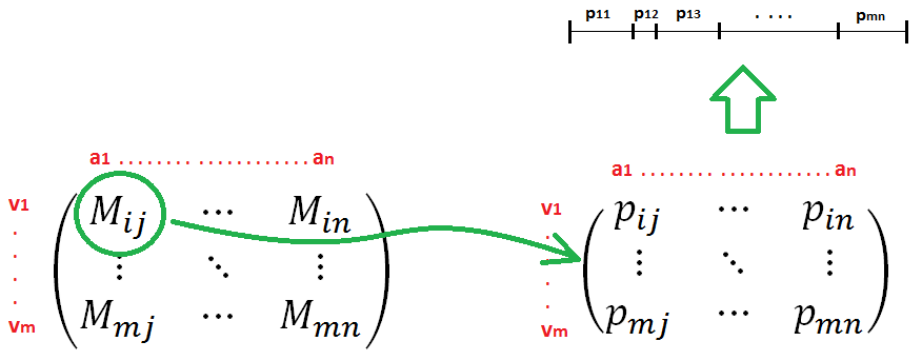
svaki element pod-matrice podijeljen je zbrojem svih elemenata pod-matrice. Prije prethodno navedenog koraka izvršeno je inkrementiranje vrijednosti pod-matrica za svaki pripadni prijelaz stanja registriran tijekom prolaska kroz ulazne vozne cikluse.



Slika 38: Matrica prijelaznih vjerojatnosti i njene pod-matrice [28]

4.5.3. Generiranje sintetičkih voznih ciklusa temeljem prijeđenog puta

Uz pomoć generatora slučajnih brojeva te matrice prijelaznih vjerojatnosti (***P***) dobivene u prethodnom poglavlju generirani su sintetički vozni ciklusi na način kako slijedi. U inicijalnom diskretnom vremenskom koraku ($k = 0$) stanja brzine i ubrzanja vozila postavljaju se na nulu ($v(0) = 0$ i $a(0) = 0$). Počevši od inicijalnog stanja i njemu pripadne podmatrice prijelaznih vjerojatnosti, određuje se stanje brzine i ubrzanja vozila u sljedećem koraku tako da se najprije uzmu u obzir sve prijelazne vjerojatnosti čija je vrijednost veća od 0 ($p_{ij} > 0$). Vrijednost svake prijelazne vjerojatnosti definira duljinu koja odgovara pod-segmentu ukupnog segmenta jedinične duljine (proizlazi iz toga da je zbroj svih elemenata pod-matrice jednak 1, Slika 39).

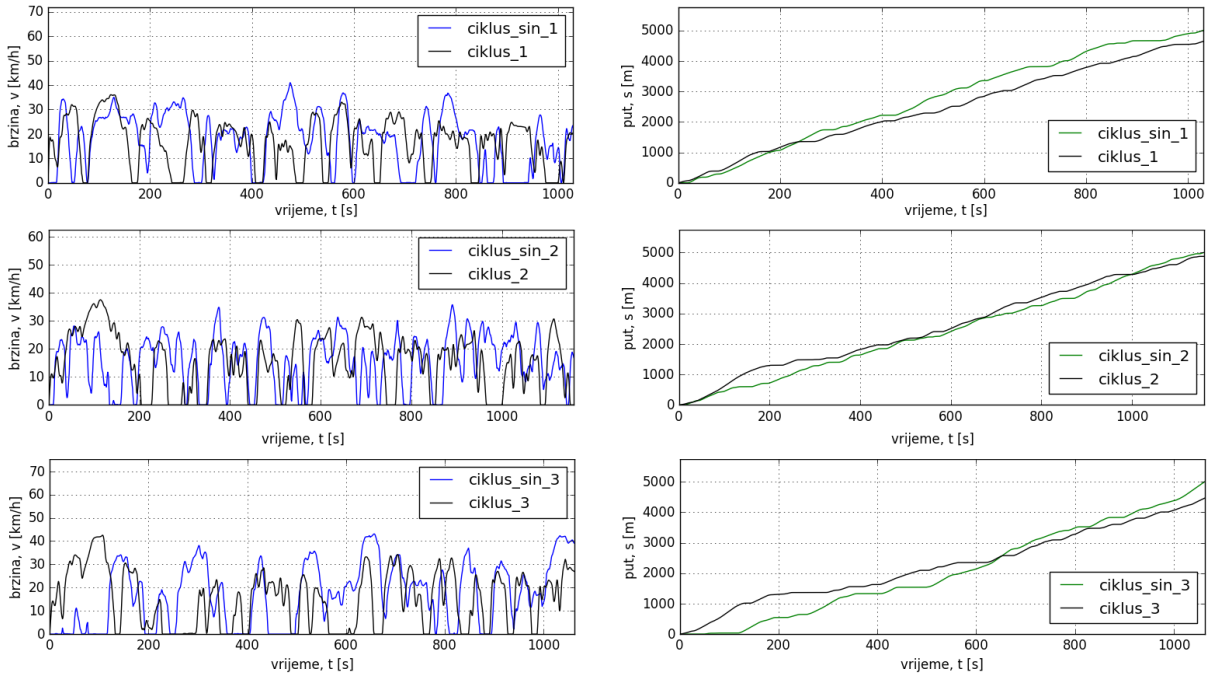


Slika 39: Projekcija prijelaznih vjerojatnosti na jedinični segment

Zatim se uz pomoć generatora slučajnih brojeva s jednolikim rasporedom i rasponom od 0 do 1 odabire pod-segment, iz kojeg dalje proizlazi stanje brzine i ubrzanja u koraku k+1 [28]. Nakon dobivenog stanja brzine i akceleracije računa se prijeđeni put:

$$s_{uk}(k) = s_0 + v(k) \cdot T \quad \textbf{(42)}$$

Postupak se iterativno ponavlja sve dok vrijednost prijeđenog puta (s_{uk}) ne prijeđe zadani iznos (definiran udaljenošću između stanica, potpoglavlje 4.3). Primjere generiranih sintetičkih voznih ciklusa prikazuje Slika 40. Kao što je vidljivo, sintetički vozni ciklusi svojim oblikom vjerno reprezentiraju realne vozne cikluse, te su dalje korišteni u simulaciji cjelokupnog sustava za dijeljenje električnih mopeda prilikom svake registracije pristupa sustavu.



Slika 40: Prikaz generiranih sintetičkih voznih ciklusa

4.6. Struktura simulacijskog algoritma

Simulacijski algoritam cjelokupnog sustava za dijeljenje e-mopeda definiran je kako slijedi (Slika 43). Najprije je kreirana baza podataka koja sadrži sve tablice navedene u potpoglavlju 2.3. Zatim su definirane klase svih elemenata sustava (korisnik, e-moped, stanica) s pripadnim im svojstvima i metodama koje se koriste tijekom simuliranja sustava. Nakon toga, inicijalizirani su sljedeći objekti:

1. Lista e-mopeda - sadrži sto objekata e-mopeda čija je inicijalna vrijednost stanja napunjenosti baterije jednaka *SoC* = 1.0.

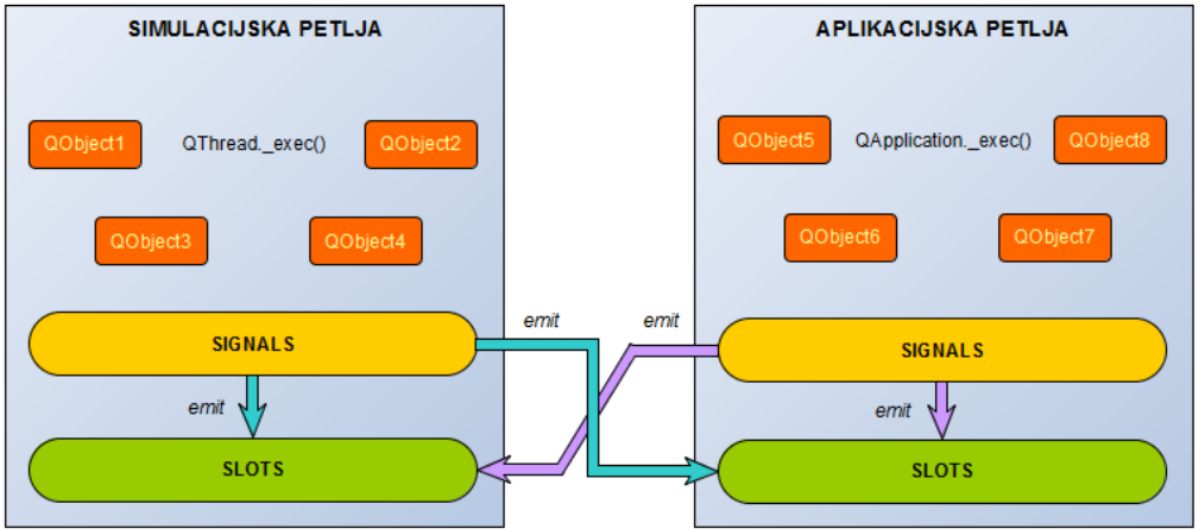
2. Lista korisnika - sadrži petsto objekata korisnika, svaki za zasebnim identifikacijskim brojem (**RFID** autorizacija), korisničkim imenom, itd.
3. Lista stanica - sadrži šesnaest objekata stanica sa svojstvima koja su definirana u potpoglavlju 4.3. Također je svakoj stanici dodijeljen određeni broj e-mopeda ovisno o njenoj lokaciji.
4. Lista slobodnih korisnika - sadrži sve slobodne korisnike u određenom diskretnom vremenskom koraku, te se njezin sadržaj mijenja tijekom simulacije sustava.
5. Lista aktivnih e-mopeda - sadrži sve objekte e-mopeda koji su aktivni u trenutnom diskretnom vremenskom koraku, te se njezin sadržaj također mijenja tijekom simulacije sustava.

Jednom kada su inicijalizirane liste potrebnih objekata pokreće se simulacijska petlja (*engl. thread*) koja je definirana kao pod-klasa *QThread* klase *PyQt* modula kako slijedi [29]:

```
from PyQt4.QtCore import QThread

class SimulacijskaPetlja(QThread):
    """
    ---------------------------------------------------------------------
    Petlja koja izvršava simulaciju cjelokupnog sustava za dijeljenje
    el. mopeda u pozadini grafičkog sučelja.
    ---------------------------------------------------------------------
    """
    # Definiranje signala
    posalji_poruku = pyqtSignal(str)
    osvjezi_status = pyqtSignal(float)
    simulacija_zavrsena = pyqtSignal(bool)
    prijava_neuspjesna = pyqtSignal(int)
    prijava_visak = pyqtSignal(int)
    pristup_sustavu = pyqtSignal(int)
    osvjezi_skutere = pyqtSignal(bool)

    # Konstruktor klase
    def __init__(self, parent):
        super(SimulacijskaPetlja, self).__init__(parent)
        self.parent              = parent
        self.BROJ_PRIJAVA        = 0
        self.PRIJAVE_VISAK       = 0
        self.PRIJAVE_NEUSPJEH    = 0
        self.lista_skutera       = copy.deepcopy(baza.podaci[0])
        self.lista_stanica       = copy.deepcopy(baza.podaci[1])
        self.lista_korisnika     = copy.deepcopy(baza.podaci[2])
        self.lkor_slob           = copy.deepcopy(self.lista_korisnika)
        self.lsk_akt             = []
        self.mapa                = Basemap(projection='merc', lat_0=0.0,
                                   lon_0=0.0, resolution='c',\
                                   llcrnrlon=18.05, llcrnrlat=42.63,
                                   urcrnrlon=18.13, urcrnrlat=42.67)
        self.zaustavljena        = False
        self.pauzirana           = False
```

```
    # Redefiniranje metode pokretanja petlje
    def run(self):
        self.adresa                 = str(self.parent.adresa.text())
        self.TSIM_TREN              = self.parent.input_tpoc.value()*3600
        self.TSIM_KRAJ              = self.parent.input_tkraj.value()*3600 - 1
        self.DELTA_TSIM             = self.parent.input_deltaT.value()
        self.DELTA_TGL              = self.DELTA_TSIM
        self.DELTA_TSK              = self.parent.input_deltaTsk.value()
        self.DELTA_TPR              = 15.0
        self.zapocniSimulacijuSustava()
```

Simulacijska petlja posjeduje posebne varijable koje su definirane kao signali (*pyqtSignal*) koji su dio tzv. *PyQt*-ovog mehanizma "Signala i Utora" (*engl. Signals and Slots*), gdje utori predstavljaju bilo koju opozivu *Python*-ovu funkciju [30]. Najbitnije svojstvo signala je da ih se može emitirati pri vlastito definiranim događajima. Emitiranjem signala ostvaruje se komunikacija između podređene simulacijske i nadređene aplikacijske petlje (Slika 41).



Slika 41: Princip rada PyQt mehanizma "Signala i Utora"

Na razini aplikacijske petlje vrše se samo izmjene u grafičkom sučelju, dok se tijekom rada simulacijske petlje odvijaju sljedeći glavni koraci:

- Ostvaruje se povezivanje s centrom vođenja te se inicijalizira baza podataka kako bi se stanja svih elemenata sustava postavila na početna.
- Za svaku stanicu generiraju se polja točnih vremena pristupa ($t_{pristupa} = [t_0, ..., t_n]$, gdje je n broj generiranih pristupa) korisnika sustavu uz pomoć metoda koje su definirane u potpoglavlju 4.2.
- Za svaki vremenski period ΔT_{sim} osvježavaju se stanja simulacije sve dok trenutno vrijeme simulacije ($t_{sim,tren}$) ne dosegne krajnju vrijednost ($t_{sim,kraj}$).

Pod osvježavanje stanja simulacije podrazumijeva se širok raspon radnji. Najprije se za određeni vremenski korak *k* za svaku stanicu provjerava da li je trenutno vrijeme ($t_{sim,tren}$) veće ili jednako prvom "izvučenom" vremenu pristupa. Ukoliko je uvjet ispunjen, registrira se pristup sustavu te se izvršavaju sljedeće operacije:

1. Ostvaruje se povezivanje sa serverom (centrom vođenja)
2. Odvija se autorizacija korisnika
3. Korisniku se dodjeljuje e-moped
4. Odabire se sljedeća destinacija i izračunava ukupna udaljenost koju e-moped mora prijeći (s_{uk})
5. Generira se sintetički vozni ciklus i skalira se krivulja nagiba ceste (Slika 31) temeljem izračunatog ukupnog puta (s_{uk})
6. Objekt e-mopeda ubacuje se u listu aktivnih e-mopeda
7. Objekt korisnika izbacuje se iz liste slobodnih korisnika
8. Vrše se ostale nužne izmjene u podacima e-mopeda i stanice koje su detaljno definirane u potpoglavlju 2.1.

Za potrebe komunikacije simulacijskog programa s centrom vođenja napisana je funkcija *posaljiPodatkeCV* koja kao ulazne argumente prima tip upita (GET ili POST), ime funkcije centra vođenja koja se želi izvršiti (npr. *autorizirajKorisnika*), te niz varijabli koje se prosljeđuju u formatu {*ključ1=vrijednost1, ključ2=vrijednost2, ...*}. Spajanje na željenu **IPv4** (*engl. Internet Protocol version 4*) adresu servera putem interneta izvedeno je korištenjem *Python*-ovog integriranog *httplib* modula kako slijedi [10]:

```
def posaljiPodatkeCV(self, funkcija, varijable, tip):
    try:
        if self.adresa == "localhost":
            konekcija = httplib.HTTPConnection(self.adresa, 8000)
        else:
            konekcija = httplib.HTTPConnection(self.adresa)
        konekcija.request(tip, "/EMS/default/" + funkcija + varijable)
        odgovor = konekcija.getresponse().read()
        return odgovor
    except:
        return None
    finally:
        konekcija.close()
```

Nakon provjera pristupa sustavu svi aktivni e-mopedi "provlače" se kroz matematički model (definiran u potpoglavlju 4.1, ali preveden u *Python* kod) te se na temelju prijeđenog puta izračunava njihova trenutna pozicija u obliku geografske širine i dužine (Slika 42).

$$\Delta s(k) = s(k) - s(k-1) \quad \textbf{(43)}$$

$$\Delta x(k) = \Delta s(k) \cdot \cos \propto \quad \textbf{(44)}$$

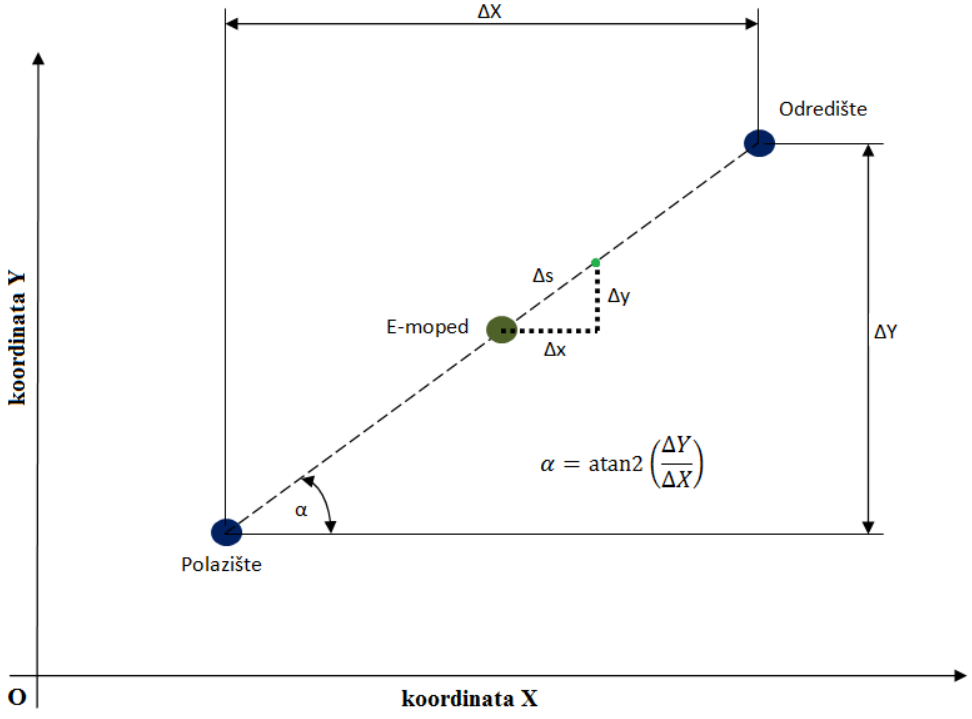
$$\Delta y(k) = \Delta s(k) \cdot \sin \propto \quad \textbf{(45)}$$

$$\Delta \varphi(k) = \frac{\Delta y(k)}{R} \cdot \frac{180}{\pi} \quad \textbf{(46)}$$

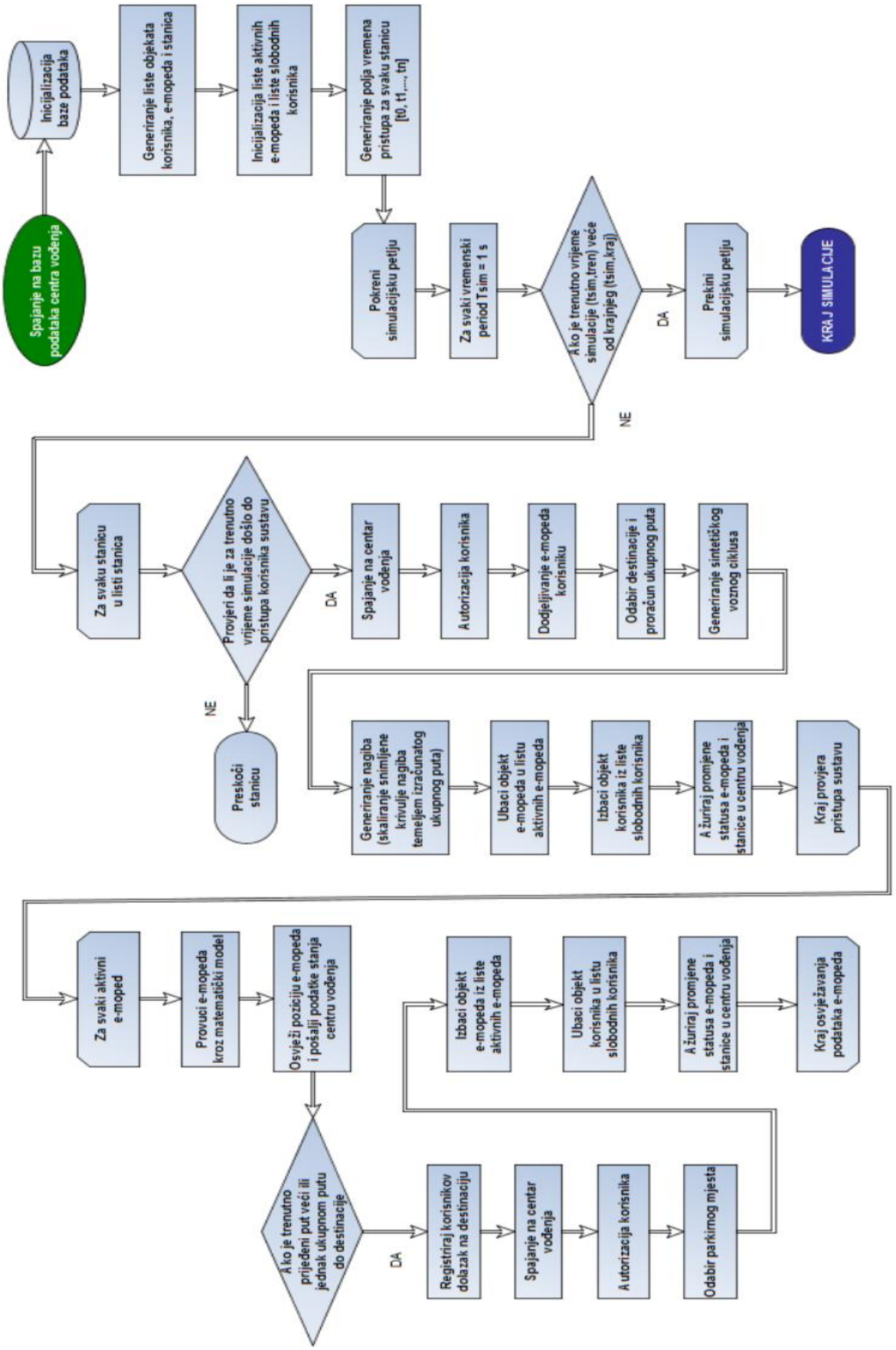
$$\Delta \lambda(k) = \frac{\Delta x(k)}{R \cos \varphi_0} \cdot \frac{180}{\pi} \quad \textbf{(47)}$$

gdje φ označava geografsku širinu, λ geografsku dužinu, R efektivni polumjer zemlje, α kut između polazne stanice i odredišta, φ_0 geografsku širinu e-mopeda u diskretnom vremenskom koraku (k-1), a s prijeđeni put od vremena polaska prema destinaciji. Ukoliko je iznos trenutno prijeđenog puta (s_{tren}) e-mopeda veći ili jednak od ukupnog puta do destinacije (s_{uk}, određenog sintentičkim voznim ciklusom), registrira se korisnikov dolazak na destinaciju, odnosno izvršavaju se sljedeće radnje: objekt e-mopeda briše se iz liste aktivnih e-mopeda, objekt korisnika dodaje se u listu slobodnih korisnika, na strani centra vođenja ažuriraju se podaci vraćanja ili ostavljanja e-mopeda.

Jednom kada su stanja e-mopeda osvježena, slijedi slanje trenutnih podataka stanja (definiranih u potpoglavlju 2.4) svih aktivnih e-mopeda prema centru vođenja, te ažuriranje promjena statusa punjenja svih stanica.



Slika 42: Prikaz proračuna putanje e-mopeda od polazišta do odredišta



Slika 43: Dijagram toka simulacijskog algoritma

5. IZVEDBA SIMULATORA KLIJENTA

5.1. Opis i struktura aplikacije

Grafičko sučelje (**GUI** - *engl. Graphical User Interface*) simulatora klijenta izrađeno je uz pomoć *PyQt* modula, odnosno jedne od najpopularnijih *Python*-ovih poveznica (*engl. binding*) s *Qt* okvirom koji podržava razne platforme (*engl. cross-platform framework*), razvijenog od strane *Riverbank Computing Limited* [31]. Glavni prozor grafičkog sučelja sastoji se od alatne trake (*engl. toolbar*), te od lijeve, desne i centralne ploče (Slika 44).

Slika 44: Prikaz glavnog prozora aplikacije simulatora klijenta

Unutar alatne trake nalaze se sljedeće naredbe:

- ***Spremi bilješke*** - omogućuje spremanje bilješki simulacije u tekstualnom (*.txt*) formatu.
- ***Spremi rezultate*** - omogućuje spremanje trenutnih rezultata simulacije u *Pickle* (.pkl) formatu. *Pickle* modul implementira fundamentalan, ali moćan algoritam za serijalizaciju i deserijalizaciju *Python*-ovih objekata, tj. njihove strukture. *"Pickling/Unpickling"* predstavljaju procese u kojima se hijerarhija *Python*-ovih objekata pretvara u niz bajtova (*engl. byte stream*) i obrnuto.
- ***Otvori rezultate*** - omogućuje otvaranje postojećih, odnosno spremljenih rezultata simulacije.

- ***Pokreni animaciju*** - pritiskom na naredbu započinje animacija sustava od početnog do krajnjeg vremena simulacije (potpoglavlje 5.2).
- **Prikaz rezultata** - pritiskom na naredbu otvara se novi prozor za iscrtavanje rezultata simulacije čiji je sadržan detaljno opisan u potpoglavlju 5.3.

Lijeva ploča (*engl. panel*) sadrži glavne opcije simulacije kao što su početno vrijeme (inicijalno postavljeno na početak dana, 00:00:00), završno vrijeme (inicijalno postavljeno na kraj dana, odnosno 24:00:00), vremenski korak simulacije (inicijalno postavljen na 1 s) i vremenski korak slanja podataka e-mopeda centru vođenja (inicijalno postavljen na 15 s), traku napretka (*engl. progress bar*), te kontrole simulacije (gumbi za pokretanje ili zaustavljanje i pauziranje ili odpauziranje simulacije).

Centralna ploča sadrži prikaz svih bilježaka simulacije čije se stanje dinamički nadopunjuje tijekom simulacije, odnosno prilikom svakog zbivanja koje zahtjeva bilježenje podataka. U bilješkama simulacije zapisana su vremena pristupa sustavu zajedno s imenima korisnika, statusi autorizacije korisnika, statusi procesa preuzimanja, vraćanja ili ostavljanja e-mopeda na stanicama, statusi konekcija s bazom podataka, te vremena polazaka e-mopeda prema odabranim destinacijama.

Unutar desne ploče nalaze se vremenski promjenjivi podaci simulacije među koje spadaju: trenutni broj registriranih pristupa sustavu, trenutni broj neuspješnih pristupa sustavu, količina slobodnih korisnika naspram ukupnog broja korisnika, te broj trenutno aktivnih e-mopeda zajedno s prikazom njihovih oznaka.

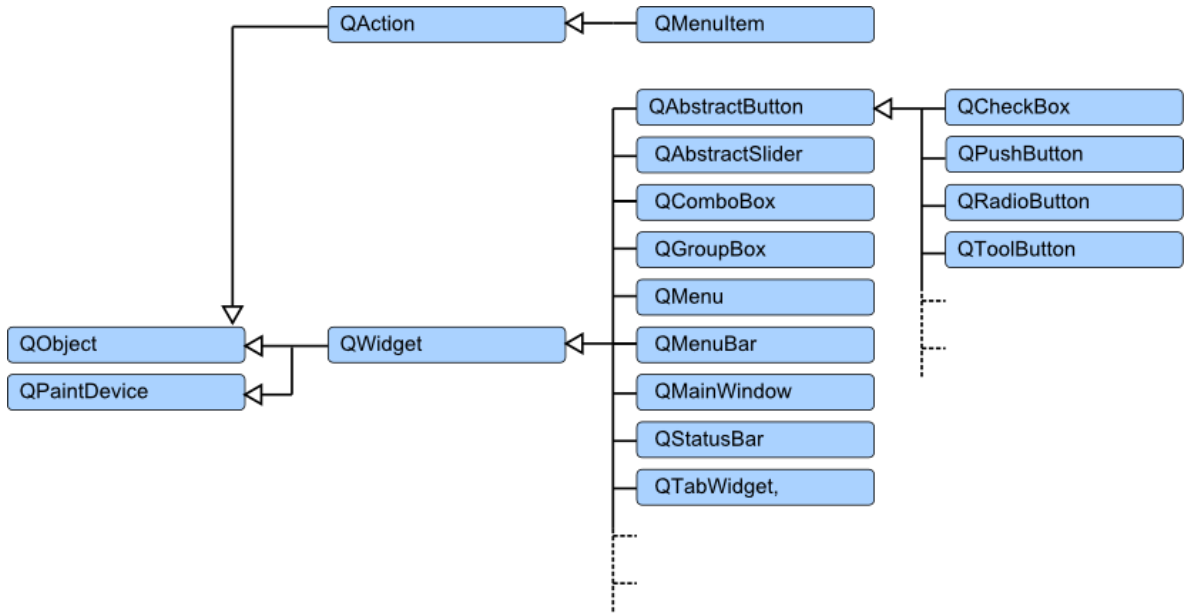
Kako bi se inicijaliziralo grafičko sučelje najprije je potrebno definirati klasu glavnog prozora kao podklasu *QMainWindow* klase (Slika 45) s ciljem nasljeđivanja svih njenih svojstava:

```
from PyQt4.QtCore import *
from PyQt4 import QtGui

STIL_DATOTEKA          = "./Resursi/CSS/stil.css"
STIL                   = open(STIL_DATOTEKA, 'r').read()

class GlavniProzor(QtGui.QMainWindow):
    def __init__(self):
        super(GlavniProzor, self).__init__()
        self.setWindowTitle('SIMULATOR KLIJENTA')
        self.setWindowIcon(QtGui.QIcon(SLIKE_FOLDER + 'logo.png'))
        self.centrirajProzor()
        self.kreirajAlatnuTraku()
        self.kreirajSadrzaj()
        self.petlja = SimulacijskaPetlja(self)
        self.rezultati = None
        self.odsimularano = False
```

gdje *centrirajProzor*, *kreirajAlatnuTraku*, *kreirajSadrzaj*, itd. predstavljaju vlastito definirane metode (funkcije definirane unutar klase) čije su količine programskih naredbi prevelike pa stoga u ovom radu njihov sadržaj neće biti prikazan.

Zatim slijedi definiranje objekta aplikacije (*QApplication*), prikaz glavnog prozora u veličini raspoloživog prostora, te na kraju pokretanje glavne aplikacijske petlje (*engl. event loop*) koja služi za registraciju sistemskih ulaza (pomak "miša", pritisak gumba, itd.).

```
import sys

aplikacija = QtGui.QApplication(sys.argv)
aplikacija.setStyleSheet(STIL)
gl_prozor = GlavniProzor()
gl_prozor.showMaximized()
sys.exit(aplikacija.exec_())
```



Slika 45: Hijerarhija upravljačkih elemenata (*engl. widgets*) PyQt modula [32]

5.2. Animacija izvršene simulacije

Animacija simulacije sustava za dijeljenje e-mopeda izrađena je uz pomoć *Python*-ovog *Pygame* modula. *Pygame* je skup *Python*-ovih modula namijenjenih za pisanje video igara. Uključuje računalnu grafiku i zvučne knjižnice (*engl. libraries*) osmišljene kako bi se koristile s *Python* programskim jezikom. Izgrađen je na bazi **SDL** (*engl. Simple DirectMedia Layer*) knjižnica, s namjerom omogućavanja razvoja računalnih igara u realnom vremenu, odnosno bez nisko-razinske mehanike *C* programskog jezika i njegovih derivata [33]. Prilikom korištenja *Pygame* modula najprije ga je potrebno inicijalizirati unutar glavne klase, te nakon

toga definirati objekte glavnog zaslona, pozadine, **FPS** (*engl. Frames per Second*) brojača i fontova:

```
import pygame, os
from pygame.locals import *

class Animacija:
    def __init__(self, sirina=1280, visina=800):
        # Inicijalizacija Pygame-a
        os.environ['SDL_VIDEO_CENTERED'] = '1'
        pygame.init()
        # Dimenzije zaslona
        self.sirina = sirina
        self.visina = visina
        # Kreacija zaslona i pozadine
        self.zaslon = pygame.display.set_mode((self.sirina, self.visina))
        self.pozadina = pygame.Surface(self.zaslon.get_size())
        self.pozadina = self.pozadina.convert()
        # Postavke zaslona
        pygame.display.set_caption("ANIMACIJA SUSTAVA")
        pygame.mouse.set_visible(True)
        # FPS brojac
        self.fpsclk = pygame.time.Clock()
        # Fontovi
        self.fontM = pygame.font.Font('freesansbold.ttf', 14)
        self.fontV = pygame.font.Font('freesansbold.ttf', 24)
```

Nakon glavne inicijalizacije potrebno je definirati glavnu petlju koja služi za osvježavanje grafike, u ovom slučaju pozicija e-mopeda na zaslonu (Slika 46). Osvježavanje grafike izvršava se u svakom diskretnom vremenski koraku *k* sve dok trenutno vrijeme animacije ne dosegne vrijednost krajnjeg vremena, a definirano je sljedećim koracima:

1. Brisanje cijeloga zaslona (ispunjavanje zaslona crnom bojom)
2. Crtanje tekstualnog sadržaja (trenutno vrijeme simulacije i broj **FPS**-a)
3. Crtanje efekata pristupa sustavu ukoliko je do njih došlo
4. Crtanje svih stanica
5. Crtanje svih e-mopeda na pozicijama u *k*-tom koraku
6. Provjera svih registriranih događaja

Unutar glavne petlje također se dohvaćaju svi registrirani događaji (pomak "miša", pritisak gumba tipkovnice, itd.), te se na temelju njih odvijaju određene akcije. Pošto za animaciju sustava korisnik ne upravlja niti jednim grafičkim objektom, definirane su samo akcije za povećanje/smanjenje **FPS**-a pritiskom na tipkala gore/dolje.

```
def pokreniGlavnuPetlju(self):
    izlaz, k, FPS, FPS_MIN, FPS_MAX = (False, 0, 60, 10, 100)
    k_max = len(self.lista_skutera[0].rezultati['lat'])
    while (izlaz == False):
        if k < k_max:
```

```
            self.pozadina.fill((0, 0, 0))
            self.nacrtajTekst(k, k_max)
            self.nacrtajEfekte(k)
            self.nacrtajStanice()
            self.nacrtajSkutere(k)
            self.zaslon.blit(self.pozadina, (0, 0))

        for event in pygame.event.get():
            if event.type == QUIT:
                izlaz = True
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    FPS += 5
                    if FPS > FPS_MAX: FPS = FPS_MAX
                elif event.key == pygame.K_DOWN:
                    FPS -= 5
                    if FPS < FPS_MIN:
                        FPS = FPS_MIN

        pygame.display.update()
        self.fpsclk.tick(FPS)
        k += 10

    pygame.display.quit()
    pygame.quit()
```

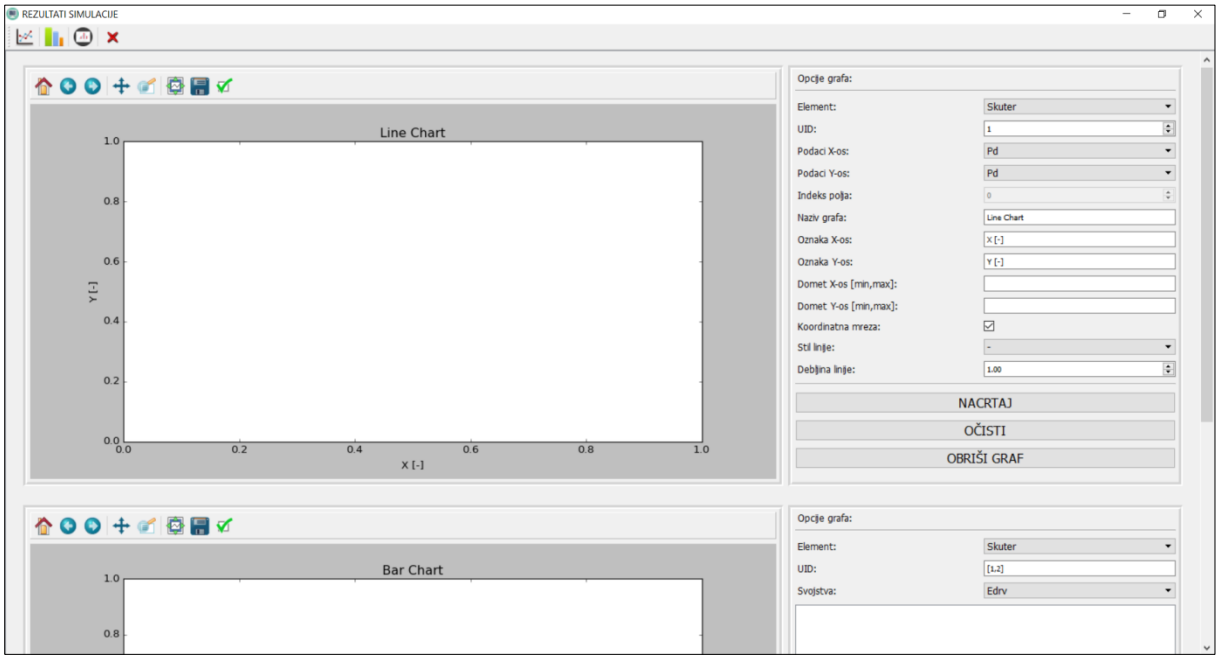
Jednom kada su definirane sve metode nužne za izvršavanje animacije, koje uključuju otvaranje rezultata simulacije, inicijalizaciju varijabli (liste e-mopeda, stanica, efekata, itd.), pokretanje glavne petlje, crtanje stanica, e-mopeda, efekata pristupa i teksta, potrebno je kreirati objekt glavnog zaslona i pokrenuti animaciju pozivanjem pripadnih metoda:

```
if __name__ == "__main__":
    """Funkcija za pokretanje animacije sustava."""
    anim = Animacija()
    anim.otvoriRezultate()
    anim.inicijalizirajVarijable()
    anim.pokreniGlavnuPetlju()
```

Slika 46: Prikaz zaslona animacije izvršene simulacije

5.3. Prikaz rezultata simulacije

Zaslon za prikaz rezultata simulacije sastoji se od alatne trake, te prostora za dodavanje određenih tipova grafova (Slika 47). Unutar alatne trake nalazi se operacija za dodavanje linijskog dijagrama (*engl. line chart*), operacija za dodavanje trakastog dijagrama (*engl. bar chart*), operacija za brisanje svih grafova s ekrana, te operacija za iscrtavanje najbitnijih dijagrama kao što su intenziteti posjećenosti stanica, usporedba troškova električne energije i fosilnog goriva, te odnos ukupno prijeđenog puta e-mopeda s promjenom stanja napunjenosti baterije.

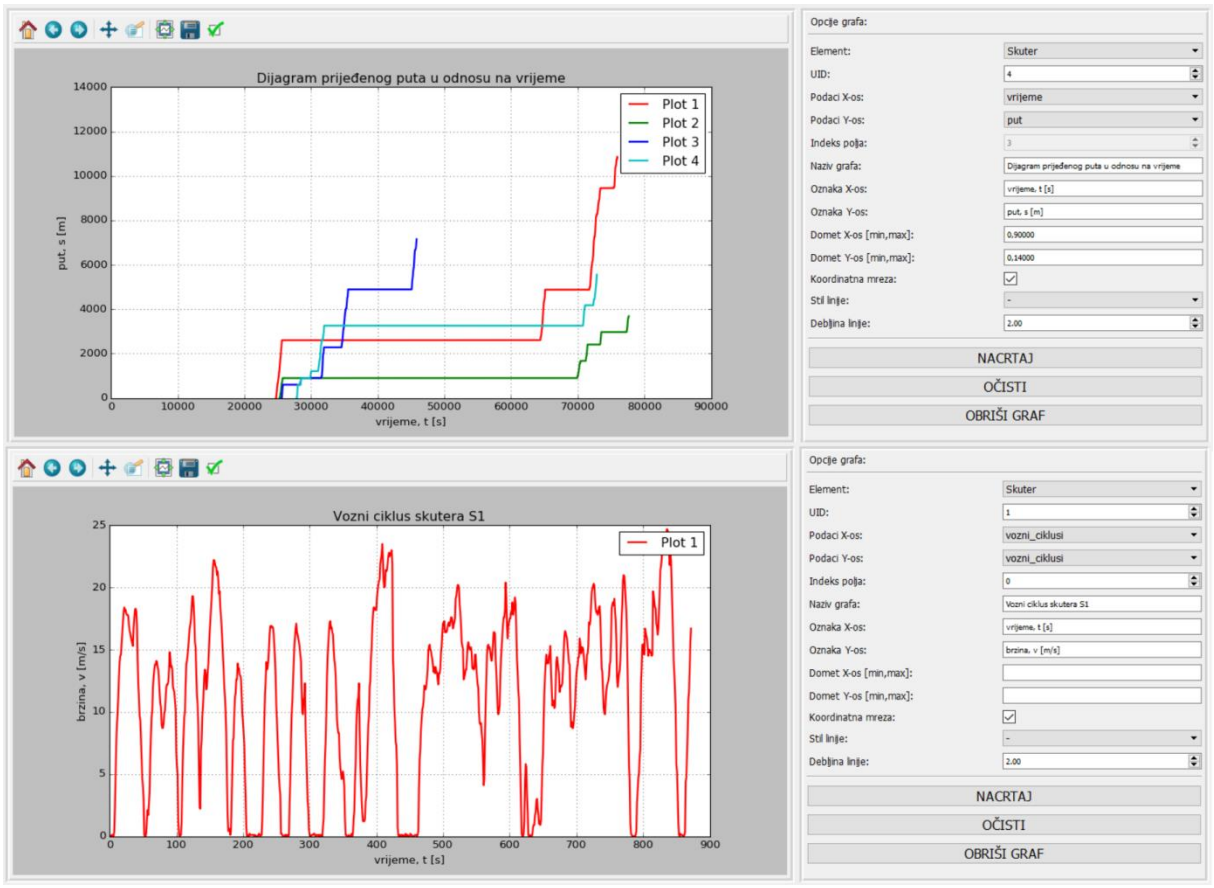


Slika 47: Zaslon za prikaz rezultata simulacije

Linijski se dijagram sastoji od lijeve i desne ploče (Slika 48). Unutar lijeve ploče nalazi se platno (*engl. figure canvas*) za iscrtavanje grafa zajedno s pripadnim mu operacijama (spremanje slike, zumiranje, itd.), dok se u desnoj ploči nalaze sve opcije grafa, zajedno s gumbima za dodavanje i brisanje njegovog sadržaja, te gumbom za brisanje cjelokupnog grafa s ekrana. Opcije grafa definirane su na sljedeći način:

- ***Element*** - odabire se e-moped ili stanica ovisno o podacima koji se žele iscrtavati.
- ***UID*** - jedinstveni identifikator odabranog elementa.
- ***Podaci X-os*** - padajući izbornik koji sadrži sve podatke odabranog elementa sustava koji su tijekom simulacije bili spremljeni.
- ***Podaci Y-os*** - padajući izbornik koji sadrži sve podatke odabranog elementa sustava koji su tijekom simulacije bili spremljeni.

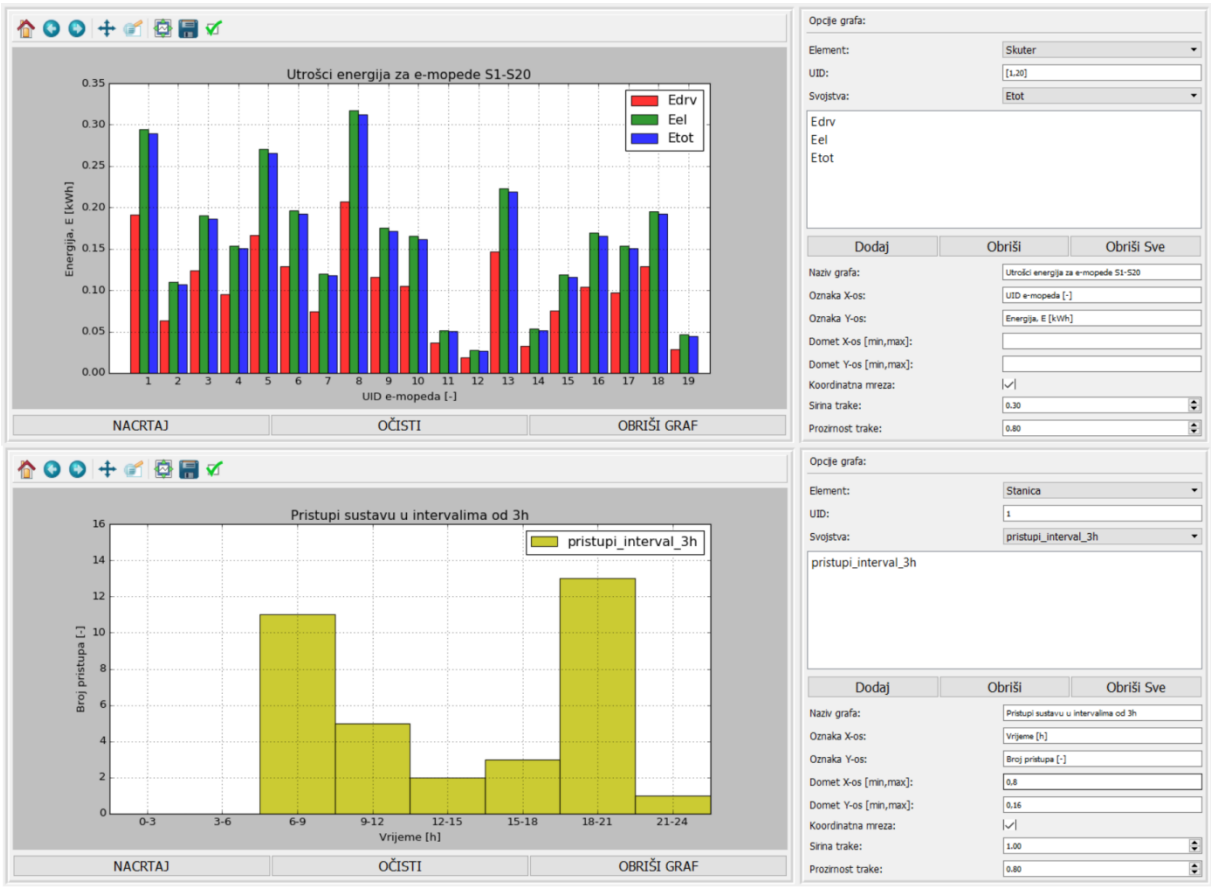
- ***Naziv grafa*** - tekstualno polje za unos naziva grafa.
- ***Oznaka X-os*** - tekstualno polje za unos oznake i dimenzije x-osi.
- ***Oznaka Y-os*** - tekstualno polje za unos oznake i dimenzije y-osi.
- ***Domet X-os*** - tekstualno polje za unos dometa (intervala) x-osi u obliku [min, max].
- ***Domet Y-os*** - tekstualno polje za unos dometa y-osi u obliku [min, max].
- ***Koordinatna mreža*** - omogućuje prikaz ili skrivanje koordinatne mreže grafa.
- ***Stil linije*** - padajući izbornik koji sadrži stilove linija za iscrtavanje željenih podataka (crta, crta-točka-crta, zvijezda, dijamant, itd.).
- ***Debljina linije*** - omogućuje unos željene debljine linije za iscrtavanje grafa.



Slika 48: Primjeri iscrtanih linijskih dijagrama

Trakasti se dijagram razlikuje od linijskog po tome što iscrtava jedne ili više spremljenih podataka za skup elemenata sustava (stanice, e-mopedi) na istom platnu (Slika 49). Također omogućuje odabir više podataka ukoliko se želi izvršiti njihova usporedba (npr. promjena stanja napunjenosti baterije u odnosu na ukupno prijeđeni put). Opcije grafa definirane su na sljedeći način:

- ***Element*** - odabire se e-moped ili stanica ovisno o podacima koji se žele iscrtavati.
- ***UID*** - interval jedinstvenih identifikatora odabranih elemenata [UID_{poc}, UID_{kraj}].
- ***Svojstva*** - odabire se jedno ili više svojstava iz padajućeg izbornika koja se žele iscrtati ili usporediti. Odabrana svojstva nalaze se u posebnom okviru. Dodaju se pritiskom na gumb "Dodaj", a brišu pritiskom na gumb "Obriši".
- ***Naziv grafa*** - tekstualno polje za unos naziva grafa.
- ***Oznaka X-os*** - tekstualno polje za unos oznake i dimenzije x-osi.
- ***Oznaka Y-os*** - tekstualno polje za unos oznake i dimenzije y-osi.
- ***Domet X-os*** - tekstualno polje za unos dometa (intervala) x-osi u obliku [min, max].
- ***Domet Y-os*** - tekstualno polje za unos dometa y-osi u obliku [min, max].
- ***Koordinatna mreža*** - omogućuje prikaz ili skrivanje koordinatne mreže grafa.
- ***Širina trake*** - odabir širine trake pojedinačnog odabranog svojstva u granicama od 0.1 do 1.0.
- ***Prozirnost trake*** - faktor prozirnosti trake u granicama od 0.1 do 1.0.



Slika 49: Primjeri iscrtanih trakastih dijagrama

6. ANALIZA REZULTATA SIMULACIJE

6.1. Procjena troškova

Procjena troškova za e-mopede izračunata je na temelju procijenjene utrošene električne energije (E_{tot}) prema cijenama određenim od strane HEP-ODS d.o.o [34]. Odabran je bijeli tarifni model za poduzetništvo, te su korištene pripadne cijene visoke tarife (VT) i niske tarife (NT) za utrošenu količinu energije od 1 kWh. U cijenama su sadržani troškovi električne energije, troškovi korištenja mreže (prijenos i distribucija), dok su troškovi mjernih mjesta korišteni samo za izračun ukupnih mjesečnih troškova (Tablica 14).

$$\Delta SoC_i = SoC_{poč,i} - SoC_{kraj,i}, \qquad i = 1, \dots, n; \quad n = br.\ e-mopeda \tag{48}$$

$$E_{tot,i} = \Delta SoC_i \cdot E_{\max} \tag{49}$$

U jednadžbi (48) i (49) $SoC_{poč}$ predstavlja početno stanje napunjenosti baterije e-mopeda, SoC_{kraj} stanje napunjenosti baterije e-mopeda nakon prijeđenog puta, a $E_{\max}$ maksimalnu količinu energije koju baterija može dati.

Tablica 14: Cijene pojedinačnih stavki za bijelu VT i NT tarifu (bez PDV-a) [34]

	VT		NT	
Stavka	Količina [kWh]	Cijena [kn/kWh]	Količina [kWh]	Cijena [kn/kWh]
Energija	1	0.71	1	0.42
Prijenos	1	0.11	1	0.05
Distribucija	1	0.24	1	0.12
Ukupno	1.06 kn/kWh		0.59 kn/kWh	
Ostali troškovi				
OIE	0.0350 kn/kWh		0.0350 kn/kWh	
Mjerno mjesto	41.30 kn/mj.		41.30 kn/mj.	

gdje su vremenski rasponi tarifa definirani kako slijedi [34]:

- zimsko računanje vremena: VT od 07-21 sata, NT od 21-07 sati
- ljetno računanje vremena: VT od 08-22 sata, NT od 22-08 sati

Kao tipični predstavnik konvencionalnih mopeda koji spadaju u B kategoriju odabran je *Piaggio FLY 50* čija snaga iznosi 3,43 kW pa će stoga prilikom usporedbe biti korišteni njegovi tehnički podaci (Tablica 15).

Tablica 15: Tehničke specifikacije mopeda Piaggio FLY 50 [5]

Agregat	Jednocilindrični, dvotaktni Piaggio Hi-PER 2
Obujam	49,4 cm^3
Provrt/hod	49 mm/39.3 mm
Snaga	4,6 KS pri 7000 o/min
Okretni moment	4,7 Nm pri 6750 o/min
Napajanje	Rasplinjač
Hlađenje	Zračno
Podmazivanje	Automatski mješač
Pokretanje	Električno i nožnim pokretačem
Mjenjač	Automatski variomat CMT
Spojka	Automatska centrifugalna
Okvir	Od čeličnih cijevi
Prednji ovjes	Teleskopska, hidraulična vilica ϕ30 mm
Stražnji ovjes	Jedan hidraulični amortizer
Prednja kočnica	Disk ϕ200 mm
Stražnja kočnica	Bubanj ϕ140 mm
Prednja guma	Bez zračnice 120/70-12"
Stražnja guma	Bez zračnice 120/70-12"
Dužina/Širina/Međuosovinski razmak	1880 mm / 735 mm / 1340 mm
Visina sjedala	785 mm
Kapacitet spremnika goriva	7,5 litre
Homologacija	EURO 2
Potrošnja goriva	cca. 2 litre / 100 km [35]

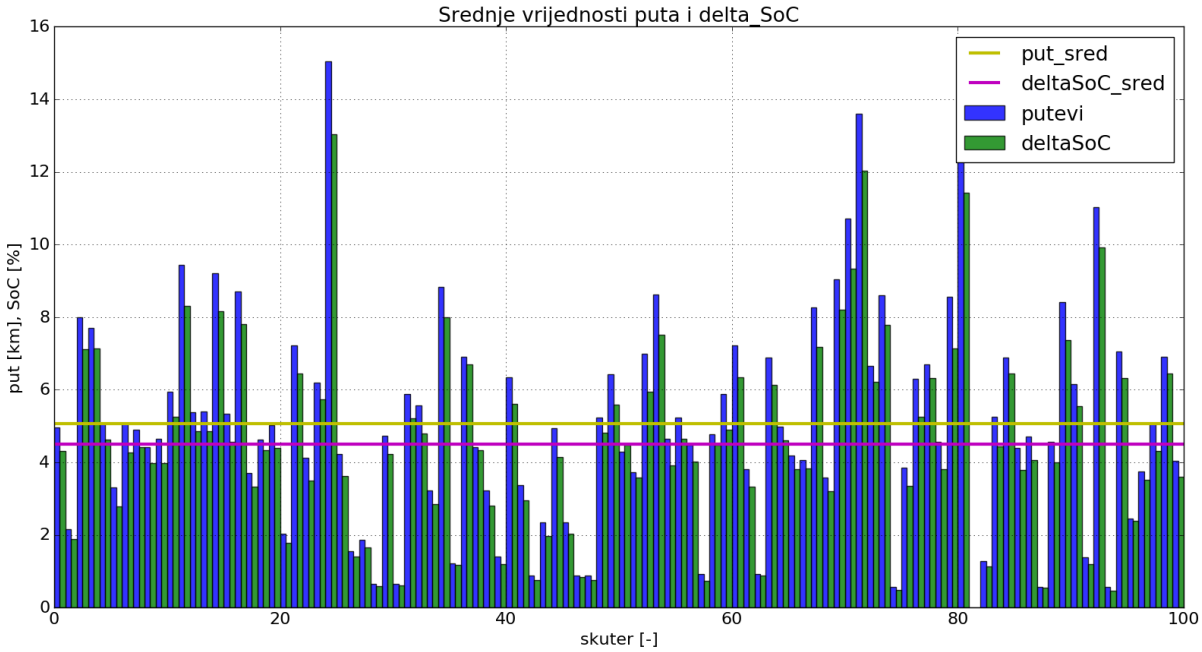
Za procijenu troškova fosilnog goriva korištena je cijena EUROSUPER 95 čiji je iznos jednak 9,64 kn/l [36].

6.2. Usporedna procjena troškova vožnje e-mopeda i konvencionalnog mopeda

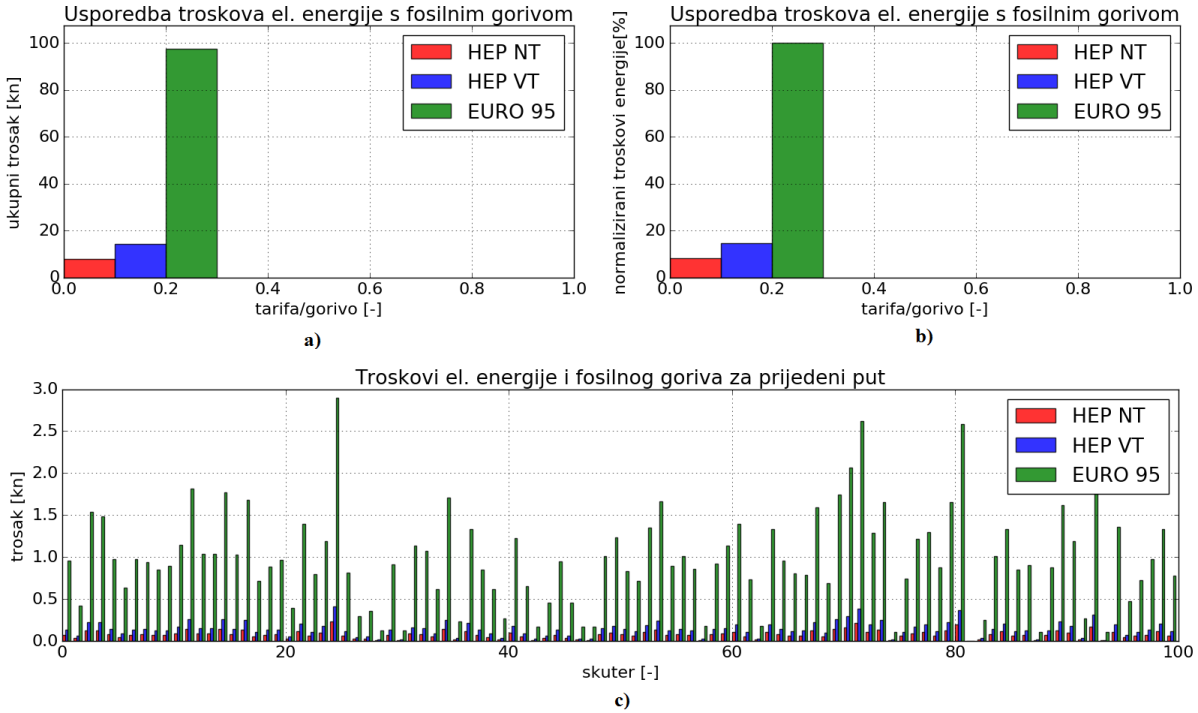
Troškovi vožnje e-mopeda izračunati su kao umnožak ukupne cijene za visoku ili nisku tarifu i ukupno utrošene električne energije ($Tr_{uk,el} = E_{tot} \cdot cijena\ tarife$), dok su troškovi konvencionalnog mopeda izračunati kao umnožak ukupno prijeđenog puta, potrošnje goriva i cijene fosilnog goriva po jednoj litri ($Tr_{uk,fg} = put_{uk} \cdot potrošnja \cdot cijena\ goriva$).

Ukupno prijeđene puteve svakog e-mopeda u kilometrima, zajedno s razlikama u stanjima napunjenosti baterija u postotcima te njihovim usrednjenim vrijednostima nakon 24-satne simulacije sustava prikazuje Slika 50, dok pojedinačne i ukupne dnevne troškove fosilnog

goriva, električne energije i troškove korištenja mreže zajedno s njihovom usporedbom prikazuje Slika 51.



Slika 50: Ukupni dnevno prijeđeni putevi i razlike u stanjima napunjenosti baterija (ΔSoC) svih e-mopeda



Slika 51: Ukupni dnevni troškovi električne energije i fosilnog goriva (a), ukupni dnevni troškovi električne energije normalizirani u odnosu na troškove fosilnog goriva (b), te pojedinačni dnevni troškovi električne energije i fosilnog goriva svih e-mopeda

Iz priloženih slika vidljivo je da troškovi fosilnog goriva znatno nadmašuju troškove električne energije, odnosno da troškovi NT iznose 8.16 %, a troškovi VT iznose 14.66 % troškova fosilnog goriva što je otprilike 7 - 12 puta manje. Uzimajući u obzir prethodno dobivene vrijednosti troškova električne energije i fosilnog goriva za 24-satnu simulaciju sustava, izračunata je približna vrijednost ukupnih mjesečnih troškova za flotu od 100 električnih i konvencionalnih mopeda, s i bez uključenih troškova mjernih mjesta (Tablica 16).

Tablica 16: Ukupni dnevni i mjesečni troškovi energije/goriva flote električnih i konvencionalnih mopeda

Stavka		Dnevni trošak [kn]	Mjesečni trošak [kn] (bez MMS-a)	Mjesečni trošak [kn] (uključujući MMS*)
Električna energija	NT	7,96 (8.16 %)**	238,86 (8.16 %)	899.66 (30.73 %)
	VT	14,30 (14.66 %)	429,14 (14.66 %)	1089.94 (37.23 %)
Fosilno gorivo		97,58	2927,52	2927,52

** MMS predstavlja ukupni trošak mjernih mjesta stanica u iznosu od 660,80 kn (41,30 kn po stanici)*

*** postoci troškova električne energije u odnosu na troškove fosilnog goriva (unutar zagrada)*

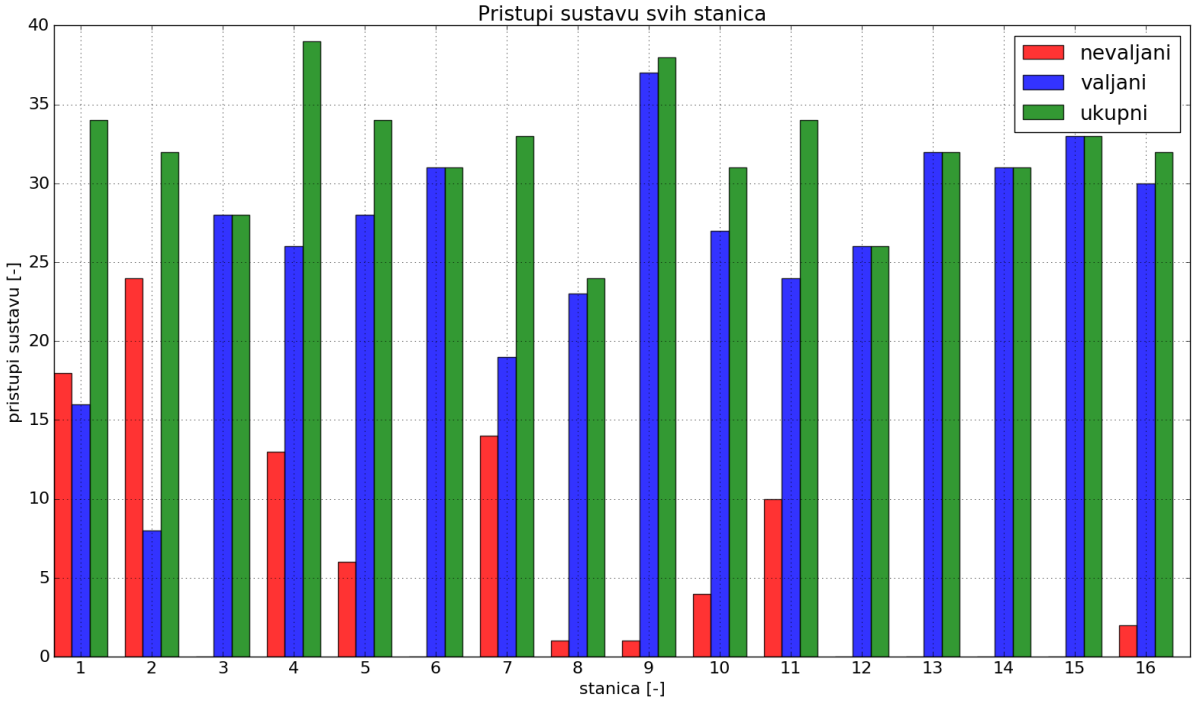
Omjer između mjesečnih troškova fosilnog goriva i troškova električne energije ($Tr_{uk,fg}/Tr_{uk,el}$) znatno je manji jednom kada su u obzir uzeti troškovi mjernih mjesta svih stanica sustava, a iznosi 325.4 % za NT, te 268.6 % za VT. Iz dobivenih rezultata proizlazi da bi se uvođenjem flote električnih mopeda naspram onih konvencionalnih znatno umanjili energetski troškovi pružatelja usluge, što bi rezultiralo i nižom cijenom njenog korištenja.

6.3. Intenziteti posjećenosti stanica

Prilikom 24-satnog simuliranja sustava za dijeljenje e-mopeda za svaku stanicu memoriran je svaki korisnički pristup neovisno o njegovoj uspješnosti. Ukoliko je korisnik bio uspješno autoriziran i stanica je posjedovala raspoloživi e-moped, broj valjanih pristupa uvećan je za 1, dok je u suprotnom uvećan broj nevaljanih pristupa. Ukupni broj pristupa stanici jednak je zbroju valjanih i nevaljanih pristupa, a njihove iznose prikazuje Slika 52.

Kao što je vidljivo iz priložene slike, određene stanice posjeduju mnoštvo nevaljanih pristupa što je rezultat njihovog visokog intenziteta posjećenosti. Ukoliko bi realni sustav pokazivao iste takve značajke, rješenje problema moglo bi se nalaziti u postavljanju drugačijeg rasporeda lokacija stanica, odnosno njihovom zgušnjavanju na područjima s najvećim intenzitetima

posjećenosti, razvoju strategije za preraspodjelu e-mopeda kojom bi se obnavljala optimalna distribucija flote među stanicama, ili pak u povećanju ukupnog broja stanica sustava.



Slika 52: Količine dnevnih pristupa sustavu za svaku stanicu

7. ZAKLJUČAK

Uvođenje usluge sustava za dijeljenje električnih mopeda u grad Dubrovnik omogućilo bi lokalnom stanovništvu i turistima jednostavan, ekonomičan i jeftin način prijevoza po širem središtu grada. Glavna prednost ovog sustava za obalne gradove upravo je u tome što koristi široko prihvaćen tip vozila, ali u električnoj verziji za koju je emisija štetnih plinova ispuštenih u okoliš jednaka nuli pa stoga doprinosi očuvanju okoliša. Korištenje tehnologija bežičnog prijenosa podataka čini sustav veoma fleksibilnim, te omogućuje neograničene mogućnosti nadogradnje sustava u pogledu razvoja dodatnih softverskih alata za obradu podataka, mobilnih aplikacija za rezervaciju e-mopeda putem mobilnih uređaja kao u otvorenim sustavima dijeljenja, i sl.

Centar vođenja, pak s druge strane pruža nadzornom timu detaljan uvid u stanje sustava u bilo kojem vremenskom trenutku, a također i jednostavno proširivanje postojeće količine elemenata sustava (e-mopeda, stanica) putem jednostavnih obrazaca za unos njihovih inicijalnih podataka u bazu podataka. Ukoliko je potrebno, svi podaci pohranjeni u bazi mogu se učiniti dostupnima potražitelju, u željenom formatu (npr. **JSON** ili **XML**), putem vlastito definiranih funkcija, odnosno **API**-a na strani servera. Cijeli je sustav isto tako veoma dobro osiguran bilo za slučajeve vandalizma (hardverske komponente s visokim stupnjem zaštite) ili za slučajeve nepoštivanja prava korištenja usluge kao što je izlazak iz dozvoljenog područja (alarmiranje nadzornog tima).

Rad cjelokupnog sustava za dijeljenje e-mopeda uspješno je realiziran i ispitan uz pomoć vlastito izrađene aplikacije simulatora klijenta koja nudi 24-satnu simulaciju sustava temeljenu na stohastičkim modelima prijave korisnika na stanicu i odabiru destinacije, zajedno sa spremanjem i prikazom svih rezultata. Za potrebe simulacija izveden je model e-mopeda koji je parametriran temeljem realističnih podataka snimljenih na modeliranom e-mopedu. Prilikom analize rezultata simulacije ustanovljeno je kako su procijenjeni ukupni troškovi utrošene električne energije e-mopeda znatno manji (otprilike 7 - 12 puta) od troškova goriva kod konvencionalnih mopeda.

Također je analizom intenziteta posjećenosti stanica dobiven uvid u moguće buduće nedostatke definiranog rasporeda stanica, te su predložena neka od rješenja (drugačiji raspored lokacija stanica, odnosno njihovo zgušnjavanje na područjima s najvećim

intenzitetima posjećenosti, razvoj strategije za preraspodjelu e-mopeda kojom bi se obnavljala optimalna distribucija flote među stanicama, povećanje ukupnog broja stanica sustava).

Kako bi simulacija sustava za dijeljenje e-mopeda bila što vjernija i bliža budućem realnom sustavu, potrebno je koristiti što više snimljenih stvarnih podataka. Neki od podataka čijim bi se snimanjem unaprijedila simulacija sustava jesu: vozni ciklusi odabranog modela e-mopeda, snimljeni nagibi cesta, intenziteti posjećenosti stanica, brojevi prijelaza iz centra u periferiju i obratno. Primjenom tako dobivenih podataka kao ulaza za algoritme stohastičkog modeliranja, iznosi određenih, u ovom radu proizvoljno definiranih koeficijenata (težine stanica, koeficijenti prijelaza iz centra u periferiju i obrnuto, itd.), bili bi statistički pouzdaniji, a generirani sintetički vozni ciklusi e-mopeda znatno vjerniji.

LITERATURA

[1] D. Gavalas, C. Konstantopoulos, G. Pantziou: "Design and Management of Vehicle Sharing Systems: A Survey of Algorithmic Approaches", Athens, Greece

[2] http://www.bikeshophub.com/contentimage/large/2014/02/toronto-bixi-martin-reis.jpg

[3] http://www.fleetsandfuels.com/wp-content/uploads/iBMW20August2012-1111.jpg

[4] Saturna: http://www.saturnagreen.com/scooter-sharing

[5] J. Deur: *Studija idejnog rješenja transportno-energetskog sustava dijeljenja e-mopeda*, Dubrovnik, 19. travnja 2016.

[6] Jakov Topić: *Projekt mehatronika i robotika*, Fakultet strojarstva i brodogradnje, Zagreb, 2016

[7] Python Software Foundation: https://docs.python.org/2/library/sqlite3.html

[8] W3Schools: http://www.w3schools.com/sql/

[9] PythonAnywhere: https://www.pythonanywhere.com

[10] Python Software Foundation: https://docs.python.org/2/library/httplib.html

[11] D. Lisjak: *Problematika odabira CMMS sustava*, Fakultet strojarstva i brodogradnje, Zagreb, 2015.

[12] Web2py: http://www.web2py.com/book

[13] https://www.google.hr/search?q=web2py&hl=hr&biw=1536&bih=735&site=imghp&source=lnms&tbm=isch&sa=X&ved=0ahUKEwipprf23aPRAhVJDCwKHaK1AuUQ_AUIBigB#hl=hr&tbm=isch&q=web2py+structure&imgrc=fNcBp-WtF4SOWM%3A

[14] https://www.google.hr/search?hl=hr&site=imghp&tbm=isch&source=hp&biw=1536&bih=735&q=aja&oq=aja&gs_l=img.3..0l10.797.1123.0.1457.3.3.0.0.0.0.104.278.2j1.3.0....0...1ac.1.64.img..0.3.277.BuyvABq9BZU#hl=hr&tbm=isch&q=ajax+web+application+model&imgrc=GmOHi55o1rzUPM%3A

[15] jQuery: http://api.jquery.com/jquery.ajax/

[16] Google Maps APIs: https://developers.google.com/maps/documentation/javascript/

[17] Govecs: http://www.govecs.de/produkte/go-s25.html

[18] B. Škugor, M. Hrgetić, J. Deur: 'GPS measurement-based road grade reconstuction with application to electric vehicle simulation and analysis', *Sustainable Development of Energy, Water and Environment Systems (SDEWES)*, 2015

[19] M. Cipek: *Modeliranje, analiza i optimalno upravljanje pogonima hibridnih električnih vozila*, Doktorski rad, Fakultet strojarstva i brodogradnje, Zagreb, 2015

[20] Rizzoni G, Guzzella L, Baumann BM.: 'Unified modeling of hybrid electric vehicle drivetrains', *IEEE/ASME Transactions on Mechatronics*, 4(3):246-257, 1999.

[21] Zou Y, Li D, Hu X.: 'Optimal Sizing and Control Strategy Design for Heavy Hybrid Electric Truck', *Mathematical Problems in Engineering*, 2012.

[22] Pourabdollah M, Murgovski N, Grauers A, Egardt B.: 'Optimal Sizing of a Parallel PHEV Powertrain', *IEEE Transactions on Vehicular Technology*, 62(6):2469-2480, 2013. doi:10.1109/TVT.2013.2240326

[23] Movable Type Scripts: http://www.movable-type.co.uk/scripts/latlong.html

[24] D. Pavković: *auditorne vježbe iz kolegija Neizrazito i digitalno upravljanje: Postupci sinteze PID regulatora*, Fakultet strojarstva i brodogradnje, Zagreb, 2014.

[25] D. Ščap, A. Jokić: *Optimiranje mehaničkih konstrukcija - teorijske osnove i primjena*, Fakultet strojarstva i brodogradnje, Zagreb, 2014.

[26] Wikipedia: https://hr.wikipedia.org/wiki/Markovljev_lanac

[27] https://www.google.hr/search?hl=hr&site=imghp&tbm=isch&source=hp&biw=1536&bih=735&q=Markov+chains&oq=Markov+chains&gs_l=img.3..0i19k1j0i30i19k1.1533.4632.0.4859.13.10.0.3.3.0.82.649.9.9.0....0...1ac.1.64.img..1.12.653...0j0i30k1.YLD5LVYt8BQ#imgrc=lAkCnWAO7ewadM%3A

[28] J. Deur, B. Škugor: 'Delivery vehicle fleet data collection, analysis and naturalistic driving cycles synthesis', *Int. J. Innovation and Sustainable Development*, Vol. 10, No. 1, 2016

[29] Mark Summerfield: *Rapid GUI Programing with Python and Qt: The Definitive Guide to PyQt Programming*, Prentice Hall, 2007/10/19

[30] ZetCode: http://zetcode.com/gui/pyqt4/

[31] Riverbank Computing: https://riverbankcomputing.com/software/pyqt/intro

[32] https://www.google.hr/search?hl=hr&site=imghp&tbm=isch&source=hp&biw=1536&bih=735&q=Markov+chains&oq=Markov+chains&gs_l=img.3..0i19k1j0i30i19k1.1533.4632.0.4859.13.10.0.3.3.0.82.649.9.9.0....0...1ac.1.64.img..1.12.653...0j0i30k1.YLD5LVYt8BQ#hl=hr&tbm=isch&q=qt+widget+hierarchy&imgrc=IIVhMRlZuDjh1M%3A

[33] Pygame: https://www.pygame.org/wiki/about

[34] HEP-ODS: http://www.hep.hr/ods/, http://mojracun.hep.hr/kalkulator/

[35] Svijet skutera: http://www.svijetskutera.com/test-piaggio-fly-4t-4v/2012/09/17/

[36] Cijene goriva: http://cijenegoriva.info/CijeneGoriva.aspx

PRILOZI

I. Matlab kod

Optimiranje parametara PID regulatora voznih ciklusa:

```
clear all; clc; format short;

global vv time dist grade KR KI KD parametri deltaT

load('Govecs_ECO_MODE.mat');
load('a_vs_v_GOVECS.mat');
load nagib_gauss4.mat;

% Zadnji rezultati (PID,ro=0.05)
K_OPT = [
14.9012   23.0316    0.0000;
16.8803   15.7506    0.0000;
16.4548   16.9435    0.0000;
16.3189   17.2968    0.0000;
39.8961   39.2083    0.0000;
15.9784   18.2682    0.0000;
16.8697   15.7933    0.0000;
];

optimiraj = input('Optimiraj? (DA=1, NE=0)\n');
if optimiraj == 1
    parametri = input('(PID=1, PI=2, P=3)\n');
else
    s = size(K_OPT);
    parametri = 4 - s(2);
end
eco_mode = input('ECO MODE: (ON=1, OFF=0)\n');
if eco_mode == 1
    tau_max = ECO_MODE_ON;
    acc_max = a_interp_eco_on;
    vv_max = v_interp_eco_on;
else
    tau max = ECO MODE OFF;
    acc_max = a_interp_eco_off;
    vv_max = v_interp_eco_off;
end
spremi = input('Spremi cikluse: (DA=1, NE=0)\n');
deltaT = 1.0;

%Parametri skutera
Af = 0.6;
g = 9.81;
r_ef = 0.2431;
Ro = 0.014;
ro_air = 1.225;
Cd = 0.9;
m_vozila_prazno = 123;
m_putnika = 75;
m_vozila = m_vozila_prazno + m_putnika;
vmax = 45.0;
```

```
options = optimset('fminsearch');
options.Display = 'iter';
options.Dignostics = 'on';
options.DiffMinChange = 1e-10;
options.MaxFunEvals = 2000;
options.MaxIter = 2000;
options.TolCon = 1e-6;
options.TolFun = 1e-6;
options.TolX = 1e-4;

if optimiraj == 1
    K_OPT = [];
    for ciklus = 1:7
        % Parametri regulatora
        if parametri == 1
            KR = 10.0;
            KI = 10.0;
            KD = 10.0;
            x0 = [KR KI KD];
        end
        if parametri == 2
            KR = 10.0;
            KI = 10.0;
            KD = 0.0;
            x0 = [KR KI];
        end
        if parametri == 3
            KR = 10.0;
            KI = 0.0;
            KD = 0.0;
            x0 = [KR];
        end
        % Optimiranje parametara
        load(strcat('ciklus', mat2str(ciklus), '.mat'));
        dist = z; grade = m;
        vv = vv *(vmax/max(vv));
        x = fminsearch('Fcilja', x0, options);
        K_OPT = [K_OPT; x];
    end
end

vv_cell = {};
acc_cell = {};
tau_cell = {};
tsim_cell = {};
vv_cell_reg = {};
acc_cell_reg = {};
tau_cell_reg = {};
tsim_cell_reg = {};

for ciklus = 1:7
    load(strcat('ciklus', mat2str(ciklus), '.mat'));
    dist = z; grade = m;
    vv = vv *(vmax/max(vv));

    sim('Vozni_ciklusi_model.mdl');

    vv_cell{ciklus} = vv_sim;
    acc_cell{ciklus} = acc;
```

```
    tau_cell{ciklus} = tau_dc;
    tsim_cell{ciklus} = tsim;
end

for ciklus = 1:7
    if parametri == 1
        KR = K_OPT(ciklus,1);
        KI = K_OPT(ciklus,2);
        KD = K_OPT(ciklus,3);
    end
    if parametri == 2
        KR = K_OPT(ciklus,1);
        KI = K_OPT(ciklus,2);
        KD = 0.0;
    end
    if parametri == 3
        KR = K_OPT(ciklus,1);
        KI = 0.0;
        KD = 0.0;
    end
    load(strcat('ciklus', mat2str(ciklus), '.mat'));
    dist = z; grade = m;
    vv = vv *(vmax/max(vv));

    sim('Vozni_ciklusi_PI_model.mdl');

    vv_cell_reg{ciklus} = vv_mj;
    acc_cell_reg{ciklus} = acc;
    tau_cell_reg{ciklus} = tau_izl;
    tsim_cell_reg{ciklus} = tsim;

    if spremi == 1
        vv = vv_mj*3.6;
        time = tsim;
        save(strcat('ciklus_reg', mat2str(ciklus), '.mat'), 'vv','time');
    end
end

for ciklus = 1:7
   vv1 = vv_cell{ciklus};
   vv2 = vv_cell_reg{ciklus};
   acc1 = acc_cell{ciklus};
   acc2 = acc_cell_reg{ciklus};
   figure('DefaultAxesFontSize',16)
   subplot(211)
   hold on
   plot(vv1, tau_cell{ciklus}, 'r*')
   plot(vv2, tau_cell_reg{ciklus}, 'b*')
   plot(v_ECO_MODE/3.6, tau_max, 'k-', 'linewidth', 2.0)
   plot(v_ECO_MODE/3.6, -tau_max, 'k-', 'linewidth', 2.0)
   title('Momenti')
   xlabel('brzina [m/s]')
   ylabel('moment [Nm]')
   legend('inicijalni ciklus', 'regulirani ciklus', 'ogranicenje momenta')
   subplot(212)
   plot(tsim_cell{ciklus}, vv1, tsim_cell{ciklus}, vv2)
   title('Brzine')
   xlabel('vrijeme [s]')
   ylabel('brzina [m/s]')
```

```
   legend('inicijalni ciklus', 'regulirani ciklus')
end
```

Funkcija cilja PID regulatora voznih ciklusa:

```
function [val] = Fcilja(x)

    global vv time dist grade KR KI KD parametri

    Kp = 0.1319;
    ro = 0.05;
    W = 1;
    if parametri == 1
        KR = x(1);
        KI = x(2);
        KD = x(3);
    end
    if parametri == 2
        KR = x(1);
        KI = x(2);
    end
    if parametri == 3
        KR = x(1);
    end
    if KR < 0 || KI < 0 || KD < 0
        val = 1e5;
        return;
    end

    sim('Vozni_ciklusi_PI_model.mdl');

    d = size(vv_mj); M = d(1);
    e = greska(W+1:M);
    u = tau_reg(W+1:M);
    u0 = vv_ref(W+1:M)/Kp;
    val = (sumsqr(e) + ro*Kp^2*sumsqr(u-u0))/(M+1);
end
```

II. Python kod

Zbog prevelike količine ukupno napisanog koda aplikacija centra vođenja i simulatora klijenta (nekoliko tisuća linija), dan je *Python* kod samo nekih od najbitnijih funkcija i klasa.

Klasa skutera:

```
import numpy as np
from scipy.io import loadmat

#=============================================================================
#                                GLOBALNE VARIJABLE
#=============================================================================
# Postavke
REGENERATIVNO_KOCENJE = False
RESURSI = "./Resursi/"

# Konstante skutera
m_vozila_praznog     = 123.0 # Masa praznog vozila [kg]
```

```
m_putnika             = 75.0 # Srednja masa putnika [kg]
m_vozila              = m_vozila_praznog + m_putnika # Uk. masa vozila [kg]
r_ef                  = 0.2431 # Efektivni radijus gume [m]
Cd                    = 0.9 # Koeficijent aerodinamičkog otpora [-]
Ro                    = 0.014 # Koeficijent trenja kotrljanja [-]
Af                    = 0.6 # Prednja površina vozila [m2]
ro_zraka              = 1.225 # Gustoća zraka [kg/m3]
g                     = 9.81 # Akceleracija slobodnog pada [m/s2]
io                    = 6.23 # Prijenosni omjer remenice [-]
eta                   = 0.95 # Stupanj korisnosti transmisije [-]
deltaT                = 1.0 # Period uzorkovanja [s]
govecs_podaci         = loadmat(RESURSI + "Govecs_ECO_MODE.mat")
vv_max                = govecs_podaci['v_ECO_MODE'][0]
tau_max               = govecs_podaci['ECO_MODE_OFF'][0]
vv_coef               = (1/3.6/r_ef)*io
tau_coef1             = (1/io)*eta**-1
tau_coef2             = (1/io)*eta
motor_podaci          = loadmat(RESURSI + "mgmap.mat")
omega_mg_lista        = motor_podaci['mapa'][:,0]
tau_mg_lista          = motor_podaci['mapa'][:,1]
c1_lista              = motor_podaci['mapa'][:,2]
c2_lista              = motor_podaci['mapa'][:,3]
c3_lista              = motor_podaci['mapa'][:,4]
sfc                   = 0.3516 # Faktor skaliranja motora [-]
smg                   = 0.0305 # Faktor skaliranja motora [-]
Pgub_mg_min           = 0.05*2000 # Minimalni gubici snage motora [W]

# Konstante baterije
Ncell                 = 50.0 # Broj baterijskih ćelija [-]
Q_bat                 = 15.9 # Ukupni kapacitet baterije [Ah]
m_bat                 = 31.5 # Masa baterije [kg]
m_cell                = m_bat/Ncell # Masa jedne ćelije baterije [kg]
Emax                  = 3.0 # Maksimalna energija baterije [kWh]
Ecell                 = Emax/Ncell # Energija jedne ćelije [kWh]
baterija_podaci       = loadmat(RESURSI + "battmap.mat")
SoC_lista             = baterija_podaci['battmap'][:,0]
Uoc_lista             = baterija_podaci['battmap'][:,1]
Rch_lista             = baterija_podaci['battmap'][:,2]
Rdch_lista            = baterija_podaci['battmap'][:,3]

#=============================================================================
#                                    KLASE
#=============================================================================
class Skuter():
    """
    -------------------------------------------------------------------------
    Klasa skutera koja sadrži sve njegove parametre i metode nužne za
    simuliranje kompletnog sustava za dijeljenje električnih skutera.
    -------------------------------------------------------------------------
    """

    def __init__(self, uid, tip, oznaka, status, soc, soh, lat, lon,
                 punjenje, parkirno_mjesto, stanica, korisnik,
                 broj_ciklusa_punjenja, datum_zadnjeg_punjenja):
        # Svojstva skutera
        self.uid = uid
        self.tip = tip
        self.oznaka = oznaka
        self.status = status
        self.soc = soc
```

```
        self.soh = soh
        self.lat = lat
        self.lon = lon
        self.punjenje = punjenje
        self.parkirno_mjesto = parkirno_mjesto
        self.stanica = stanica
        self.korisnik = korisnik
        self.broj_ciklusa_punjenja = broj_ciklusa_punjenja
        self.datum_zadnjeg_punjenja = datum_zadnjeg_punjenja
        self.ukupni_put = 0.0
        self.put_tren = 0.0
        self.temperatura = 0.0
        self.napon = 0.0
        self.struja = 0.0
        self.greske = None
        self.korak = 0
        # Rezultati simulacije
        self.rezultati = {}
        self.rezultati['brzina'] = []
        self.rezultati['put'] = []
        self.rezultati['vrijeme'] = []
        self.rezultati['akceleracija'] = []
        self.rezultati['nagib'] = []
        self.rezultati['tau_kotaca'] = []
        self.rezultati['omega_kotaca'] = []
        self.rezultati['Pd'] = []
        self.rezultati['omega_mg'] = []
        self.rezultati['tau_mg'] = []
        self.rezultati['tau_mg_limit'] = []
        self.rezultati['Pel'] = []
        self.rezultati['Pmeh'] = []
        self.rezultati['Pgub_mg'] = []
        self.rezultati['Pgub_bat'] = []
        self.rezultati['SoC'] = []
        self.rezultati['SoH'] = []
        self.rezultati['dsoct'] = []
        self.rezultati['napon'] = []
        self.rezultati['struja'] = []
        self.rezultati['vozni_ciklusi'] = []
        self.rezultati['nagib_ciklusi'] = []
        self.rezultati['lat'] = []
        self.rezultati['lon'] = []

    def simulirajModel(self, vv, dvdt, putevi, nagibi):
        # Jednadzbe gibanja
        alfa = np.deg2rad(tablica1D(putevi, nagibi, self.put_tren))
        self.tau_kotaca = r_ef*(m_vozila*(dvdt + g*(np.sin(alfa) +
                          np.tanh(vv)*Ro*np.cos(alfa))) +
                          0.5*ro_zraka*Cd*Af*vv**2)
        self.omega_kotaca = vv/r_ef
        Pd = self.tau_kotaca*self.omega_kotaca
        # Motor/generator
        self.omega_mg = io*self.omega_kotaca
        self.tau_mg = self.tau_kotaca*(1/io)*eta**(-
                      np.sign(self.tau_kotaca))
        # Izracun snaga i limita
        tau_koef = (1/io)*eta**(-
                   np.sign(self.tau_kotaca))*np.sign(self.tau_mg)
        tau_mg_limit = tablica1D(vv_max, tau_max, vv)*tau_koef
        if abs(self.tau_mg) > abs(tau_mg_limit):
```

```
            self.Pmeh = tau_mg_limit*self.omega_mg
        else:
            self.Pmeh = self.tau_mg*self.omega_mg
        # Willans - gubici
        self.Pgub_mg = izracunajGubitkeWillans(self.omega_mg, self.tau_mg)
        # Snaga motora
        Pel = self.Pmeh + self.Pgub_mg
        self.Pel = Pel
        if REGENERATIVNO_KOCENJE == False:
            if self.Pel < 0.0:
                self.Pel = 0.0
        # Baterija
        Qmax = Q_bat*3600
        Pcell = self.Pel/Ncell
        soc = self.soc/100
        Uoc = tablica1D(SoC_lista, Uoc_lista, soc)
        Rch = tablica1D(SoC_lista, Rch_lista, soc)
        Rdch = tablica1D(SoC_lista, Rdch_lista, soc)
        if Pcell >= 0:
            R = Rdch
        else:
            R = Rch
        self.struja = Pel/Ncell/Uoc
        self.napon = Uoc
        Pgub_bat = Ncell*R*(Pcell/Uoc)**2
        dsoct = (np.sqrt(Uoc**2 - 4*R*Pcell) - Uoc)/(2*Qmax*R)
        soc += dsoct*deltaT
        self.soc = soc*100
        # Rezultati
        self.rezultati['brzina'].append(vv)
        self.rezultati['put'].append(self.ukupni_put + self.put_tren)
        self.rezultati['akceleracija'].append(dvdt)
        self.rezultati['nagib'].append(alfa)
        self.rezultati['tau_kotaca'].append(self.tau_kotaca)
        self.rezultati['omega_kotaca'].append(self.omega_kotaca)
        self.rezultati['Pd'].append(Pd)
        self.rezultati['tau_mg'].append(self.tau_mg)
        self.rezultati['omega_mg'].append(self.omega_mg)
        self.rezultati['tau_mg_limit'].append(tau_mg_limit)
        self.rezultati['Pel'].append(self.Pel)
        self.rezultati['Pmeh'].append(self.Pmeh)
        self.rezultati['Pgub_mg'].append(self.Pgub_mg)
        self.rezultati['Pgub_bat'].append(Pgub_bat)
        self.rezultati['SoC'].append(self.soc)
        self.rezultati['SoH'].append(self.soh)
        self.rezultati['dsoct'].append(dsoct)
        self.rezultati['struja'].append(self.struja)
        self.rezultati['napon'].append(self.napon)
```

Proračun matrice prijelaznih vjerojatnosti i generiranje sintetičkih voznih ciklusa:

```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
import random

#=========================================================================
#                              GLOBALNE VARIJABLE
#=========================================================================
# Postavke
```

```
UKLJUCI_AKCELERACIJE = True

# Konstante
vc_lista             = [loadmat(RESURSI + "ciklus_reg{0}.mat".format(k))
                        for k in range(1,8)]
amax_podaci          = loadmat(RESURSI + "a_vs_v_GOVECS.mat")
vmax_lista           = amax_podaci["v_interp_eco_off"][0]
amax_lista           = amax_podaci["a_interp_eco_off"][0]
v_rez                = 0.1
a_rez                = 0.4
v_max                = 45.0
v_retci              = np.arange(0.0, v_max, v_rez)
a_retci              = np.arange(-2.0, 2.0 + a_rez, a_rez)
dimv                 = np.size(v_retci)
dima                 = np.size(a_retci)
deltaT               = 1.0

#=============================================================================
#                                   FUNKCIJE
#=============================================================================
def normalizirajRetkeMatrice(M):
    """
    ---------------------------------------------------------------------------
    Normalizira retke matrice te vraća novu matricu vjerojatnosti.
    ---------------------------------------------------------------------------
    """
    dim = np.shape(M)
    Mn = np.zeros(dim)
    for i in range(0, dim[0]):
        redak = M[i,:]
        suma = np.sum(redak)
        rnovi = redak/suma
        Mn[i,:] = rnovi
    return Mn

def normalizirajMatricu(M):
    """
    ---------------------------------------------------------------------------
    Normalizira podmatrice ulazne matrice te vraća novu matricu
    vjerojatnosti.
    ---------------------------------------------------------------------------
    """
    dim = np.shape(M)
    Mn = np.zeros(dim)
    for i in np.arange(0, dim[0]):
        for j in np.arange(0, dim[1]):
            suma = np.sum(M[i,j])
            if suma == 0:
                suma = 1.0
            Mn[i,j] = M[i,j]/suma
    return Mn

def generirajVozniCiklus(put_uk):
    put_suma = 0.0
    a0, v0, tsim = (0.0, 0.0, 0.0)
    idx_ak = np.abs(a_retci - a0).argmin()
    idx_vk = np.abs(v_retci - v0).argmin()
    brzine = [v0]
    akceleracije = [a0]
    vremena = [tsim]
```

```
    putevi = [put_suma]
    while (put_suma < put_uk):
        tsim += deltaT
        vv = v_retci[idx_vk]/3.6
        put_suma += vv*deltaT
        putevi.append(put_suma)
        if UKLJUCI_AKCELERACIJE == True:
            p = Pn[idx_vk, idx_ak]
            p_arr = []
            for i in range(0, dimv):
                for j in range(0, dima):
                    if p[i,j] > 0:
                        p_arr.append((p[i,j], i, j))
            rand = random.random()
            suma = 0.0
            for i,v in enumerate(p_arr):
                suma += v[0]
                if rand <= suma:
                    idx_vk = v[1]
                    idx_ak = v[2]
                    break
            vremena.append(tsim)
            brzine.append(v_retci[idx_vk])
            akceleracije.append(a_retci[idx_ak])
        else:
            p = Pn[idx_vk]
            p_arr = [(v, i) for i,v in enumerate(p) if v>0]
            rand = random.random()
            suma = 0.0
            for i,v in enumerate(p_arr):
                suma += v[0]
                if rand <= suma:
                    idx_vk = v[1]
                    break
            vremena.append(tsim)
            brzine.append(v_retci[idx_vk])
    return (np.asarray(vremena), np.asarray(putevi), np.asarray(brzine))

#==============================================================================
#                                GLAVNI PROGRAM
#==============================================================================
# Inicijalizacija matrice prijelaznih vjerojatnosti
if UKLJUCI_AKCELERACIJE == True:
    P = np.zeros((dimv, dima, dimv, dima))
else:
    P = np.eye(dimv)

# Generiranje matrice P iz snimljenih voznih ciklusa
for vc in vc_lista:
    vremena = np.array([t[0] for t in vc['time']])
    brzine = np.array([vv[0] for vv in vc['vv']])
    dvdt = np.diff(brzine)/np.diff(vremena)
    for i,v in enumerate(vremena):
        try:
            v_k = brzine[i]
            a_k = dvdt[i]
            v_k2 = brzine[i+1]
            a_k2 = dvdt[i+1]
            idx_v1 = np.abs(v_retci - v_k).argmin()
            idx_v2 = np.abs(v_retci - v_k2).argmin()
```

```
            idx_a1 = np.abs(a_retci - a_k).argmin()
            idx_a2 = np.abs(a_retci - a_k2).argmin()
            if UKLJUCI_AKCELERACIJE:
                P[idx_v1, idx_a1, idx_v2, idx_a2] += 1
            else:
                P[idx_v1, idx_v2] += 1
        except:
            pass

# Normalizacija matrice P
if UKLJUCI_AKCELERACIJE:
    Pn = normalizirajMatricu(P)
else:
    Pn = normalizirajRetkeMatrice(P)
```

Generiranje pristupa korisnika na stanicu za centar i periferiju:

```
import numpy as np
import matplotlib.pyplot as plt
import random

#=========================================================================
#                              GLOBALNE VARIJABLE
#=========================================================================
vremena             = np.arange(0, 24*60, 15, dtype=float)
mi1, sigma1, o1     = (8.0*60, 2.0*60, 2.0)
mi2, sigma2, o2     = (14.0*60, 4.0*60, 1.0)
mi3, sigma3, o3     = (20.0*60, 2.0*60, 1.5)
mi4, sigma4, o4     = (8.0*60, 2.0*60, 1.5)
mi5, sigma5, o5     = (14.0*60, 4.0*60, 1.0)
mi6, sigma6, o6     = (20.0*60, 2.0*60, 2.0)

#=========================================================================
#                                  FUNKCIJE
#=========================================================================
def generirajVjerojatnosti(centar):
    vjerojatnosti = np.zeros(np.shape(vremena), dtype=object)
    for i,v in enumerate(vremena):
        if centar == True:
            lamda = 1/gauss_c[i]
        else:
            lamda = 1/gauss_p[i]
        x = np.array([k for k in np.arange(100)])
        E_x = lamda*np.exp(-lamda*x)
        suma = 0.0
        E_novi = []
        for e in E_x:
            E_novi.append(e)
            suma += e
            if suma >= 0.99:
                break
        ostatak = 1.0 - suma
        E_novi = np.asarray(E_novi)
        vjerojatnosti[i] = E_novi + ostatak/np.size(E_novi)
    return vjerojatnosti

def generirajPristupeSustavu(centar):
    pristupi = np.zeros(np.shape(vremena))
    for i,v in enumerate(vremena):
        if centar == True:
```

```
            E = vjerojatnosti_c[i]
        else:
            E = vjerojatnosti_p[i]
        rand = random.random()
        suma = 0.0
        for j in np.arange(np.size(E)):
            suma += E[j]
            if rand < suma:
                br_korisnika = j
                break
        pristupi[i] = br_korisnika
    pristupi_int = []
    idx, suma = (0, 0.0)
    for i,p in enumerate(pristupi):
        idx += 1
        suma += p
        if idx == 12:
            pristupi_int.append(suma)
            idx, suma = (0, 0.0)
    return (pristupi, np.asarray(pristupi_int))

#=============================================================================
#                                GLAVNI PROGRAM
#=============================================================================
# Generiranje oblika normalnih razdioba (očekivanja)
gauss1 = (1.0/sigma1*np.sqrt(2*np.pi))*np.exp(-(vremena -
          mi1)**2/(2*sigma1**2))
gauss2 = (1.0/sigma2*np.sqrt(2*np.pi))*np.exp(-(vremena -
          mi2)**2/(2*sigma2**2))
gauss3 = (1.0/sigma3*np.sqrt(2*np.pi))*np.exp(-(vremena -
          mi3)**2/(2*sigma3**2))
gauss4 = (1.0/sigma4*np.sqrt(2*np.pi))*np.exp(-(vremena -
          mi4)**2/(2*sigma4**2))
gauss5 = (1.0/sigma5*np.sqrt(2*np.pi))*np.exp(-(vremena -
          mi5)**2/(2*sigma5**2))
gauss6 = (1.0/sigma6*np.sqrt(2*np.pi))*np.exp(-(vremena -
          mi6)**2/(2*sigma6**2))
f1 = o1/np.max(gauss1)
f2 = o2/np.max(gauss2)
f3 = o3/np.max(gauss3)
f4 = o4/np.max(gauss4)
f5 = o5/np.max(gauss5)
f6 = o6/np.max(gauss6)
gauss1 *= f1
gauss2 *= f2
gauss3 *= f3
gauss4 *= f4
gauss5 *= f5
gauss6 *= f6
gauss_p = gauss1 + gauss2 + gauss3
gauss_c = gauss4 + gauss5 + gauss6

# Generiranje vjerojatnosti pristupa
vjerojatnosti_c = generirajVjerojatnosti(True)
vjerojatnosti_p = generirajVjerojatnosti(False)
```

Odabir destinacije korisnika:

```
import numpy as np
from scipy.io import loadmat
import random

#=========================================================================
#                              GLOBALNE VARIJABLE
#=========================================================================
# Konstante
Rz                  = 6378137e-3 # Efektivni radijus zemlje [km]
p2c, c2p            = (0.6, 0.7) # Periferija - centar koeficijenti

#=========================================================================
#                                  FUNKCIJE
#=========================================================================
def izracunajUdaljenost(p1, p2, metoda='haversin'):
    """
    -----------------------------------------------------------------------
    Izračun udaljenosti između dvaju točaka čije su koordinate dane u
    obliku geografske širine i dužine uz pomoć raznih metoda čije
    preciznosti variraju.
    -----------------------------------------------------------------------
    """
    lista_metoda = ['haversin', 'SLC', 'pythagoras']
    if metoda in lista_metoda:
        lat1, lon1 = p1
        lat2, lon2 = p2
        fi1 = np.deg2rad(lat1)
        fi2 = np.deg2rad(lat2)
        delta_fi = np.deg2rad((lat2-lat1))
        delta_lambda = np.deg2rad((lon2-lon1))
        if metoda == "haversin":
            a = np.sin(delta_fi/2)**2 +
                np.cos(fi1)*np.cos(fi2)*np.sin(delta_lambda/2)**2
            c = 2*np.arctan2(np.sqrt(a), np.sqrt(1-a))
            d = Rz*c
            return d
        elif metoda == "SLC": # SLC = Spherical Law of Cosines
            d = np.arccos(np.sin(fi1)*np.sin(fi2) +
                np.cos(fi1)*np.cos(fi2)*np.cos(delta_lambda))*Rz
            return d
        elif metoda == "pythagoras":
            x = delta_lambda*np.cos((fi1+fi2)/2);
            y = fi2 - fi1
            d = np.sqrt(x**2 + y**2)*Rz
            return d
    else:
        raise Exception("Nije odabrana ispravna metoda.")

def odaberiDestinaciju(poc_stanica, lista_stanica):
    """
    -----------------------------------------------------------------------
    Generiranje vjerojatnosti odabira određene stanice kao moguće
    destinacije i njen odabir.
    -----------------------------------------------------------------------
    """
    lamda = 1.0
    # Lista mogućih destinacija (centar/periferija)
    rand = random.random()
```

```
    lista_stanica = [s for s in lista_stanica if s.uid != poc_stanica.uid]
    if poc_stanica.pripadaCentru() == True:
        if rand <= c2p:
            lista_stanica = [s for s in lista_stanica if
                             s.pripadaCentru()== False]
        else:
            lista_stanica = [s for s in lista_stanica if
                             s.pripadaCentru() == True]
    else:
        if rand <= p2c:
            lista_stanica = [s for s in lista_stanica if
                             s.pripadaCentru() == True]
        else:
            lista_stanica = [s for s in lista_stanica if
                             s.pripadaCentru() == False]
    # Izracunaj udaljenost između trenutne i svih ostalih stanica
    udaljenosti = []
    for stanica in lista_stanica:
        p1 = poc_stanica.dohvatiGeoKoordinate()
        p2 = stanica.dohvatiGeoKoordinate()
        d = izracunajUdaljenost(p1, p2)
        udaljenosti.append((d, stanica.uid, stanica.tezina))
    # Sortiraj prema udaljenostima
    dsort = sorted(udaljenosti, key=lambda x:x[0], reverse=False)
    # Generiraj težine bazirane na udaljenostima
    dtezine = []
    for i,v in enumerate(dsort):
        e = np.exp(-lamda*dsort[i][0])*dsort[i][2]
        uid = dsort[i][1]
        dtezine.append((e, uid))
    # Generiraj vjerojatnosti
    tezine = np.array([t[0] for t in dtezine])
    dnorm = tezine/np.sum(tezine)
    dtezine = [(dnorm[i], dtezine[i][1]) for i in range(0,len(dnorm))]
    # Odaberi destinaciju
    rand = random.random()
    suma = 0.0
    for i,v in enumerate(dtezine):
        suma += v[0]
        if rand <= suma:
            odabir = v
            break
    for s in lista_stanica:
        if s.uid == odabir[1]:
            destinacija = s
            break
    p1 = poc_stanica.dohvatiGeoKoordinate()
    p2 = destinacija.dohvatiGeoKoordinate()
    udaljenost = izracunajUdaljenost(p1, p2)*1000
    return (destinacija, udaljenost)
```